VULNERABILITY ANALYSIS OF THE PLAYER COMMAND AND CONTROL PROTOCOL

THESIS

John T. Hagen, Civ, USAF

AFIT/GCO/ENG/12-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/12-16

VULNERABILITY ANALYSIS OF THE PLAYER COMMAND AND CONTROL PROTOCOL

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

John T Hagen, BS

Civ, USAF

June 2012

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GCO/ENG/12-16

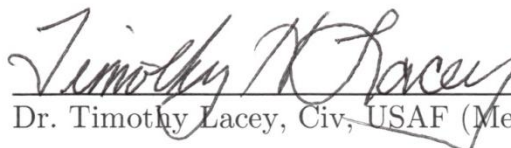# VULNERABILITY ANALYSIS OF THE PLAYER COMMAND AND CONTROL PROTOCOL

John T. Hagen, BS

Civ, USAF

Approved:

_____
Dr. Barry Mullins, Civ, USAF (Chairman)

12 Apr 12
Date

_____
Dr. Timothy Lacey, Civ, USAF (Member)

4/12/2012
Date

_____
Maj Todd Andel, Ph.D., USAF (Member)

16 APR 2012
Date

AFIT/GCO/ENG/12-16

Abstract

The Player project is an open-source effort providing a control interface specification and software framework for abstracting robot hardware. This research presents five exploits that compromise vulnerabilities in Player's command and control protocol. The attacks exploit weaknesses in the ARP, IP, TCP and Player protocols to compromise the confidentially, integrity, and availability of communication between a Player client and server. The attacks assume a laptop is connected in promiscuous mode to the same Ethernet hub as the client and server in order to sniff all network traffic between them. This work also demonstrates that Internet Protocol Security (IPsec) is capable of mitigating the vulnerabilities discovered in Player's command and control protocol. Experimental results show that all five exploits are successful when Player communication is unprotected but are defeated when IPsec Authentication Header (AH) and Encapsulating Security Protocol (ESP) are deployed together (AH+ESP) in transport mode. A cost function is defined to synthesize three distinct scalar costs (exploit success, CPU utilization, and network load) into a single scalar output that can be used to compare the different defense protocols provided by IPsec. Results from this cost function show that in a scenario when exploits are likely, IPsec AH+ESP is the preferred defense protocol because of its relatively low CPU and network overhead and ability to defeat the exploits implemented in this research by authenticating and encrypting the transport and application layers. Performance data reveals that for the Overo Earth embedded system running a TI OMAP3530 processor at 720MHz, IPsec AH+ESP increases CPU utilization by 0.52% and the network load by 22.9Kbps (64.3% increase).

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Barry Mullins, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. I would, also, like to thank my sponsor, Mr. Juan Carbonell, from the Air Force Research Labs for both the support and latitude provided to me in this endeavor.

John T. Hagen

Table of Contents

List of Figures

List of Tables

List of Equations

VULNERABILITY ANALYSIS OF THE PLAYER COMMAND AND CONTROL PROTOCOL

I. Introduction

Remotely piloted vehicles have transformed the way the U.S. military conducts operations. The advantages of such systems to perform dull, dirty, or dangerous missions are also being realized in the civilian sector. For these systems to consistently perform at their maximum potential, security must be considered when designing the communication protocols that define how these systems are remotely controlled. If compromised, these systems could lead to the loss of confidential information or the loss of control of the system. In the worst-case, the command and control system could be completely taken over by a malicious adversary, which could lead to the loss of technology or life. Because of these potential consequences, analysis of the security of the communication protocols used to remotely pilot vehicles is vital.

Player is an open-source command and control application that provides interfaces to remotely control and read sensor data from a mobile robot [GSV00]. Because it is open-source and widely used in the academic realm, it is an appropriate candidate for studying the security of command and control protocols of remotely piloted vehicles. Furthermore, the Player community has not published works discussing the security of Player, thus there is a need for work in this area.

1.1 Objectives

This thesis focuses on one part of the overall security concerns for remotely piloted vehicles: vulnerability analysis of Player's command and control protocol. The research goals of this thesis are:

1) Demonstrate the vulnerability of the Player protocol to network attacks;

2) Demonstrate the effectiveness of IPsec to secure the Player protocol;

1

3) Quantify the cost of IPsec to secure the Player protocol.

It is hypothesized that vulnerabilities in Player's command and control protocol will be discovered that allow exploits to compromise communication. Additionally, it is expected that IPsec will mitigate these vulnerabilities and allow for secure Player command and control. Finally, it is hypothesized that the system will consume additional resources when employing IPsec but that the system will maintain proper functionality.

1.2 Implications

By analyzing Player's vulnerability to attack, the community is made aware of any discovered weaknesses in the Player protocol and possible countermeasures that ensure secure deployment. The methodology outlined in this thesis can be used to analyze the security of other Player-like command and control applications. Because there is concern that mobile devices do not possess the necessary resources to protect communication, this research determines if modern mobile devices have sufficient resources to protect Player-like command and control with IPsec.

1.3 Thesis Overview

Chapter 2 provides a literature review of network security for client-server applications and a detailed description of the Player application. Related works in the field of performance analysis of security protocols are also included. Chapter 3 defines the research goals of this thesis and the methodology used to accomplish these goals. Chapter 4 presents the results and analysis of the data collected in this thesis. Finally, Chapter 5 concludes by summarizing the results and significance of this work and identifying areas for future work.

II. Literature Review

2.1 Chapter Overview

This chapter provides a review of the foundational literature for the research detailed in this thesis. The reader should be familiar with computer networking and the Internet stack. The concepts described in this chapter are implemented and extended in this thesis to accomplish the research goals.

Section 2.2 defines network security and the security model used in this thesis. Section 2.3 provides background into the current mechanisms used to secure client-server applications. Section 2.4 describes the Player project and the protocol it uses to communicate. Section 2.5 details the architecture of the physical robotic platform selected for this research. Section 2.6 reviews published network attacks against client-server applications as well as techniques that have been developed to mitigate these attacks. Section 2.7 discusses the results from works related to the field of mobile security. Section 2.8 details the novelty of this research, and Section 2.9 provides a summary of this chapter.

2.2 Network Security Fundamentals

This section defines the term network security within the context of this thesis. The cryptographic community often uses the variable names Alice and Bob to represent two parties who wish to communicate securely. Because security is difficult to define without an adversary with ill intent, the community uses the character, Eve (short for eavesdrop), to represent an adversary who can read all messages that Alice and Bob communicate to each other. The malicious character, Mallory, is not only capable of reading all messages communicated between Alice and Bob, but can additionally modify these messages, replay old messages, or create new messages. This illustration casts *network security* as the scenario in which Alice and Bob wish to communicate securely even in the presence of Mallory. Figure 1 depicts an example scenario

in which Mallory impersonates Alice and Bob by sending messages that falsely claim to originate from either Alice or Bob.

"Hi, I'm Alice."   "Hi, I'm Alice."

"Hi, I'm Bob."   "Hi, I'm Bob."

Alice   **Mallory**   **Bob**
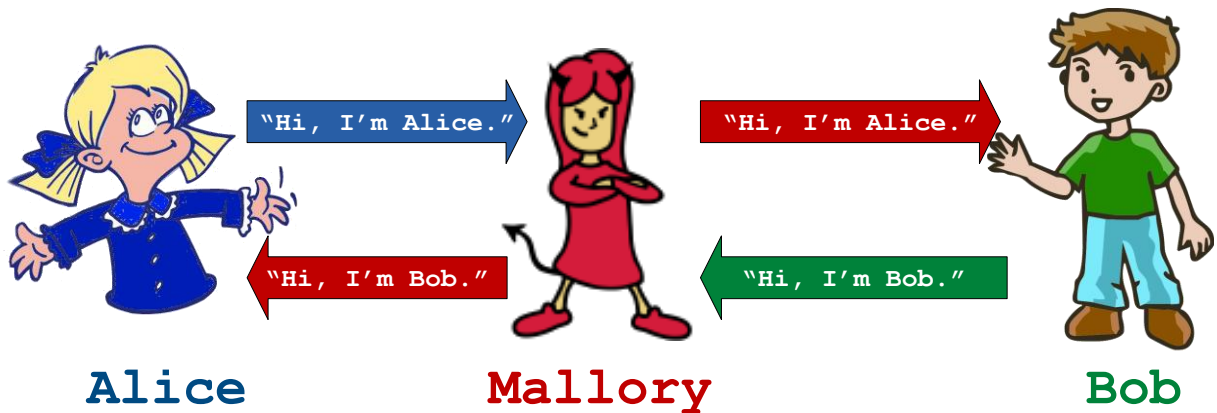
Figure 1. Mallory Impersonates Alice and Bob.

The confidentiality, integrity, and availability (CIA) security model selected for this research decomposes security into three principles. *Confidentiality* is the principle that knowledge of the communication between Alice and Bob is restricted to only Alice and Bob. *Integrity* includes two parts, data integrity and origin integrity. *Data integrity* stipulates that any modification Mallory makes to a message is detectable. *Origin integrity*, also called *authentication*, is the principle that Alice can verify that messages that claim to have originated from Bob could only have originated from Bob. *Availability* is the principle that Alice and Bob are able to send each other messages when needed and cannot be blocked by Mallory's actions [Bhi96].

2.3 Client-Server Application Security

This section describes the most widely used mechanisms for protecting the principles of the CIA security model for client-server applications that communicate over the Internet. To do so, the client-server architecture is first defined, then popular security protocols that operate in various layers in the Internet stack are evaluated. Understanding how these security

protocols protect client-server applications is crucial to the selection of a subset of these defensive protocols for study in Chapter 3.

2.3.1 *Client-Server Architecture.* The *client-server architecture* leverages cooperative processing capabilities through the use of networks to split the processing performed by the client and the server, while still presenting a single logical service to the user. A *server* is a process that exists to provide services to one or more clients [GuT95]. A *client* is a process that requests and receives information from a server. Servers do not initiate contact with clients but instead listen for requests from clients. Once a client makes a request, the server processes and services the request. When the data is returned to the client, the client operates on the data and presents it to the application, which may include a graphical user interface (GUI) for user interaction [GuT95]. Mobile client-server computing is an extension of this architecture for mobile environments. What distinguishes it from classical, fixed-connection, computing is the fact that clients can change locations and typically have higher resource constraints [JHE99].

2.3.2 *Security at the Application Layer.* *Pretty Good Privacy* (PGP) provides cryptographic confidentiality and authentication for data files and email messages [Cal07]. Confidentiality is provided using a combination of public-key and symmetric-key encryption in which symmetric session keys are generated using public-key material. Authentication is handled differently in PGP than in other Public Key Infrastructures (PKI). Rather than a top-down certificate authority, used in SSL, PGP uses a bottom-up *web of trust* model. In this model, users exchange and accumulate keys with other users they designate as trusted entities [Cal07]. Keys signed by trusted entities or signed by multiple partially trusted entities are deemed legitimate. Keys signed by unknown entities or by a single partially trusted entity are not deemed legitimate. By using a decentralized approach, PGP has the advantage of being resilient to single-node failure but scales poorly [Ger00].

5

Secure Shell (SSH) is a security protocol that creates an authenticated, encrypted channel between two networked systems. SSH is built on a client-server architecture for which the server is responsible for accepting or rejecting incoming connections to the host system. Users run SSH client programs to authenticate with, and make requests of, the SSH server [Ope09]. SSH provides authentication, encryption, and integrity through a variety of encryption algorithms, hashing algorithms, and authentication options. SSH differs from PGP in that PGP typically secures a single file or email at a time, while SSH secures an ongoing session [BSB05].

Secure Sockets Layer (SSL) was developed by Netscape Communication Corporation to provide security and privacy to Internet communication [Rsa11]. While the protocol is application-independent, it is optimized for HTTP. SSL provides encryption, client-server authentication, and message authentication codes (MAC) at the application layer. The SSL handshake is made up of a server authentication phase and an optional client authentication phase. The most recent implementation of SSL is Transport Layer Security (TLS), version 1.2. TLS 1.2 adds support for the Advanced Encryption Standard (AES) as well as the Secure Hash Algorithm 2 (SHA-2) family. Rather than the bottom-up model used in PGP, SSL employs a top-down certificate authority (CA) to authenticate clients and servers. A *certificate authority* is a trusted third party that issues digital certificates that bind a name to a public-key [Die08].

2.3.3 Security at the Transport Layer. There are currently no widely-adopted security protocols deployed for the transport layer of the Internet stack. *Obfuscated TCP* (ObsTCP) is a rejected draft for the Internet Engineering Task Force (IETF) that proposed opportunistic encryption at the transport layer. Encryption in ObsTCP is opportunistic because if either side does not support ObsTCP, the connection falls back to normal, unencrypted TCP. In comparison to SSL, ObsTCP is designed to provide faster encryption without protection from a man-in-the-middle (MITM) attack. Because it operates at the transport layer, any application

6

layer protocol can utilize ObsTCP without modification. After the IETF rejected the proposal, the draft was removed from the IETF and development of the project ceased [Obf12].

Tcpcrypt is a recently proposed transport layer security protocol that provides opportunistic encryption and optional authentication. It is similar to the failed ObsTCP IETF draft except that it also provides hooks for applications to provide authentication, which protect against MITM attacks. Because tcpcrypt operates in the transport layer, it has the advantage over application layer security protocols that it can authenticate the TCP header and be used to protect any application with less modification [BHH10]. Tcpcrypt is currently under development by a group of researchers at Stanford University, lead by Andrea Bittau [Bit12]. The IETF is currently reviewing a draft of the tcpcrypt protocol [BBH11].

2.3.4 Security at the Network Layer. Internet Protocol Security (IPsec) is a network layer security protocol designed to mitigate many of the security weaknesses inherent to the Internet Protocol (IP). These weaknesses are used in practice to perform IP spoofing, session hijacking, man-in-the-middle (MITM), and denial of service (DoS) attacks. IPsec is designed to complement upper-layer protocols (e.g., TCP) such that they do not have to be modified in order to employ its protections. Security associations (SA) are used by IPsec to define the security parameters that allow two hosts to communicate securely. A SA is uniquely identified by an IP destination address, Security Parameter Index (SPI), and a security protocol [Kim07].

As shown in Figure 2, IPsec operates in two modes: transport and tunnel. In transport mode, only the payload (typically a TCP segment) is protected. In tunnel mode, the entire IP datagram is protected and encapsulated in a new IP packet. Tunnel mode is often used to create virtual private networks (VPN).
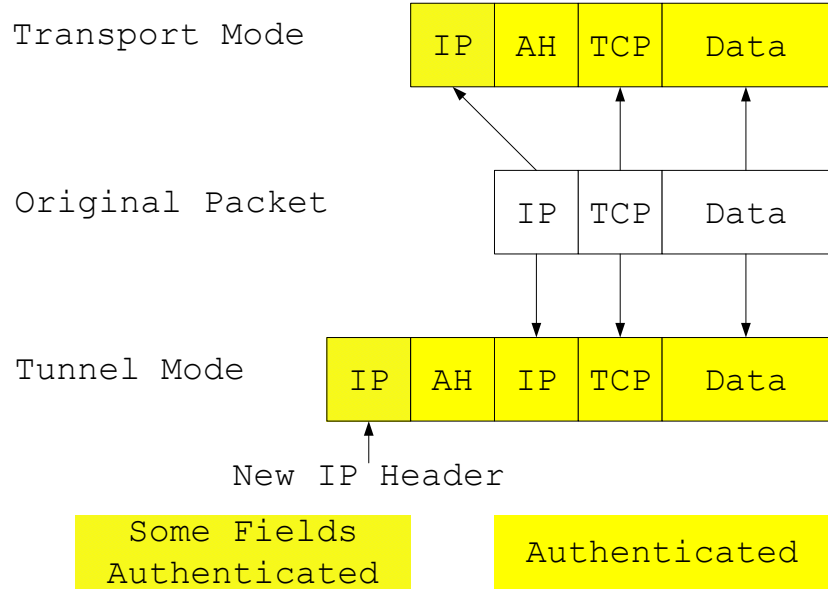
Figure 2. Comparison of IPsec AH Transport and Tunnel Modes [Ips11].

IPsec defines two protocols, the Authentication Header (AH) protocol, which provides authentication and integrity and the Encapsulating Security Payload (ESP) protocol, which provides encryption and optional authentication and integrity [KeS05]. AH inserts a header between the network and transport layers to provide authentication and integrity to most of the IP header, as well as all of the transport and application layers. AH uses a cryptographic hash-based message authentication code (HMAC) with a secret key to provide authentication and integrity [Ken05]. Figure 3 highlights the fields of an IP packet that are protected by AH. MD5 and SHA-1 are the typical HMAC's used by IPsec implementations, though both have proven weaknesses that make them undesirable for long-term use [BlC06, WYY05]. SHA-2, a family of HMACs that succeed SHA-1, currently has no published weaknesses threatening its secure usage. For this reason, the National Institute of Standards (NIST) recommends the use of SHA-2 until the results of the SHA-3 competition are finalized in late 2012 [Nis08].
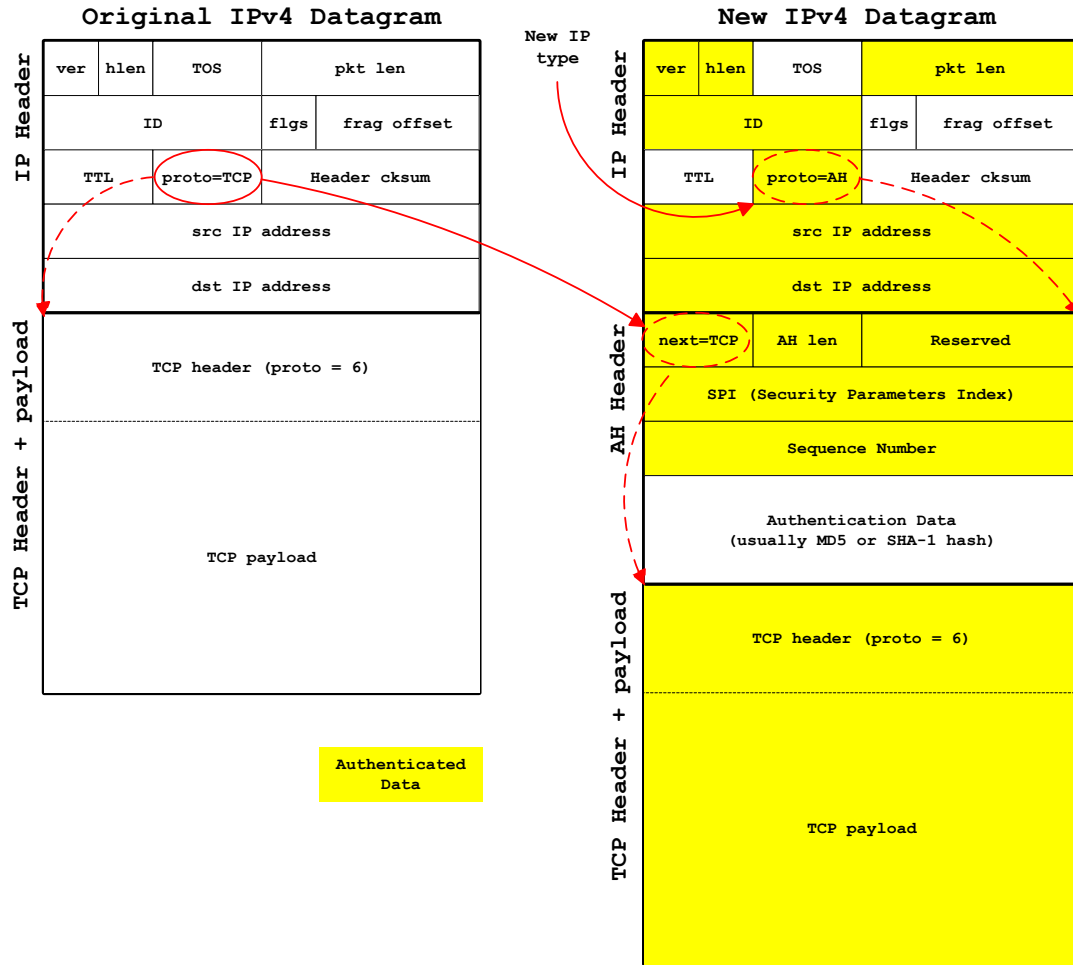
Figure 3. IPsec AH in Transport Mode [Fri05]

ESP, the second protocol defined in IPsec, inserts a small header between the network and transport layers to encapsulate and encrypt the IP payload using a secret key, as shown in Figure 4 [Ken05a]. The most common encryption algorithms used are DES, 3DES and AES, though DES is considered insecure because of its small key size [Rsa11a]. Figure 4 highlights the fields that are encrypted by ESP in transport mode and outlines the optional authentication of the ESP header and payload.
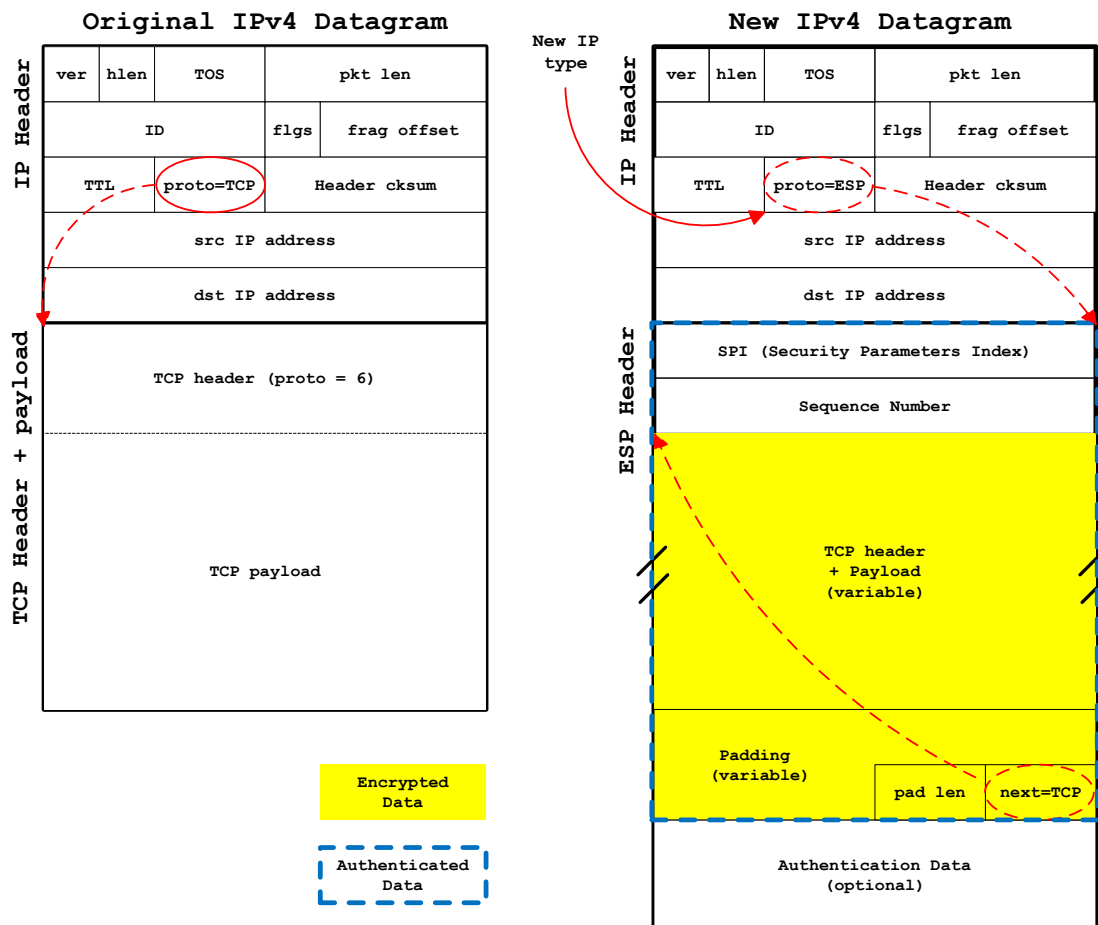
Figure 4. IPsec ESP in Transport Mode [Fri05]

Bellovin demonstrates chosen plaintext, fragmentation, and session hijacking attacks against ESP when authentication is not used [Bel96]. Bellovin's chosen plaintext attack requires 2^{32} packets, which he accomplishes in under 10 minutes on 100BaseT Ethernet. Paterson and Yau demonstrate several additional real-world attacks against the Linux kernel using ESP when authentication is not used [PaY06]. Results from their work show their attack implementation requires on average 2^{15} packets and executes in under 3 minutes. Both papers conclude that encryption without authentication is dangerous and should not be used because it is still possible for an attacker to modify and inject data into the encrypted channel without detection.

2.4 Player Project

This section describes the purpose and implementation of Player, the command and control application studied in this research. Development of the experiments detailed in Chapter 3 relies heavily on a thorough understanding of how Player functions and communicates.

2.4.1 Player. The Player project is an open-source effort providing a control interface specification and software framework for abstracting robot hardware. The project name derives from Shakespeare's *As You Like It*: "All the world's a stage, And all the men and women merely *players*" [GSV00]. Player provides simple and complete control over the physical sensors and actuators of a mobile agent. Player can handle virtually any number of clients allowing for a network of robots to communicate and cooperate. It is written to be language and platform independent, though client plug-ins currently only exist for C++, Java, and Python.

Player is designed as a client-server architecture in which robots running Player server receive commands and send status information to controlling Player clients. Robots that participate in the command and control of other robots can accomplish this by running both a Player client and a Player server locally. An example scenario where multiple Player clients and servers are used is shown in Figure 5 [GVS01]. In this scenario, Player servers (P) running on robots and other sensing devices send data to Player clients (C) that map, log, and graphically display this data. Certain robots run Player clients locally, allowing them to process data from other servers to make determinations about their surroundings.
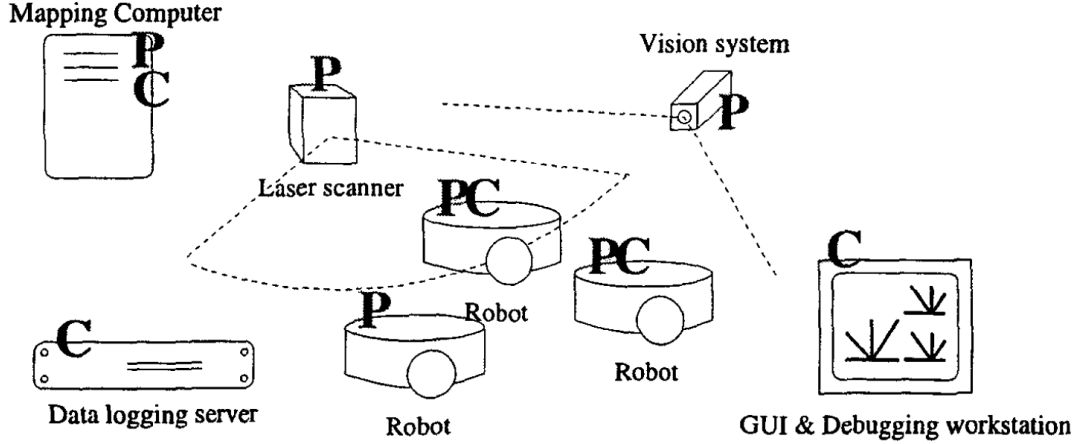
Figure 5. Scenario of Networked Player Servers (P) and Clients (C) [GVS01]

A Player server listens on TCP port 6665 for incoming client connections [GSV00]. The server provides interfaces to clients through a series of abstractions depicted in Figure 6. Available interfaces depend on the hardware that is present in the robot and include services for controlling two-dimensional (2D) position and robot peripherals such as sonar or grippers. Clients subscribe to one or more of these interfaces, allowing them to issue commands to and receive data from the robot.
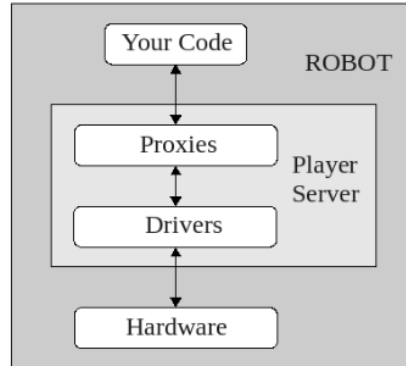
Figure 6. Player Server Architecture [Owe10]

2.4.2 Stage. Stage is a 2D robot simulation environment built to interface with Player and demonstrate robot behavior. The project name is also derived from the same line in Shakespeare's *As You Like It* [GSV00]. Stage virtualizes the physical robot from Player (Figure 7) so that a Player robot can be studied in a simulated environment. It interfaces with Player

12

the same way a physical robot does by receiving commands and moving a simulated robot. The simulated robot passes sensor data back from its virtual environment to the Player client.
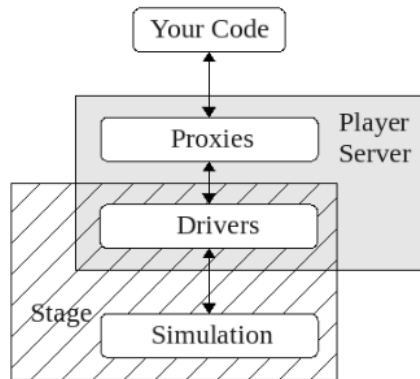
Figure 7. Stage Architecture [Owe10]

*2.4.3 Player Network Protocol.* The Player protocol defines how messages between the client and server are formatted in order to access the interfaces that Player supports. Understanding the protocol is difficult because the documentation on the official website is both outdated and incomplete. The official manual states: "Todo: -More verbose documentation on this library, including the protocol" [Pla11]. Because the protocol details are essential to this research, the following sections outline a design recovery of the Player protocol used in v3.0.2.

2.4.3.1 XDR. Player uses the IETF standard, External Data Representation (XDR), to encode messages that are passed between the client and server. XDR is well-documented so understanding the encoding of a message is straightforward once the underlying structures have been determined. All data types are encoded using 4-byte alignment (e.g., 8-bit characters are padded to fit a 4-byte cell) and transmitted in network order (big-endian) [Fre00]. As an example, the encoding for the IEEE single-precision floating point number is pictured in Figure 8.

byte 0	byte 1	byte 2	byte 3
s	exp	fractional	

←1→ ←——8 bits——→ ←————————23 bits————————→
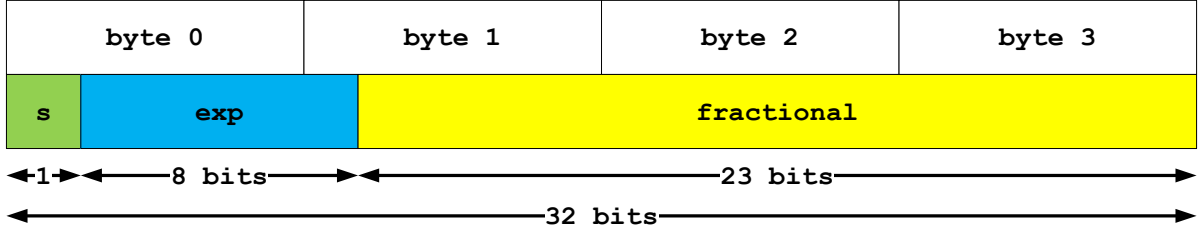
←————————————32 bits————————————→

Figure 8. XDR Representation of Floating Point [Fre00]

      2.4.3.2 Player Message Header. Any network attack against Player needs the ability to interpret the message header. Every XDR-encoded Player message begins with a 40 byte, fixed-length header, an example of which is shown in Table 1. The header structure is determined using both static analysis of the source code and dynamic packet examination of a Player client v3.0.2 communicating with Stage to obtain the raw data shown in the 4-byte XDR cells column. Enumeration values in the first column are taken from the Player source code. The first four XDR cells (host, robot, interface, index) address the particular server and interface for which the message is destined. The following two XDR cells (type, subtype) specify the contents of the message payload. A timestamp is included to synchronize the client and server. The sequence number field is no longer used because Player employs TCP for its transport service. Finally, the size field includes the length of the message payload.

Table 1. Example Player Header

				4-byte XDR cells				Type	Name	Value	Description
<-- 4*10 = 40-byte header -->	player_msghdr	player_dev addr	int int int int	00	00	00	00	uint32_t	host	0	server address (unused by client)
				00	01	1A	09	uint32_t	robot	6665	robot identifier
				00	00	00	04	uint16_t	interface	4	PLAYER_POSITION2D_CODE
				00	00	00	00	uint16_t	index	0	device identifier
		int int double int int int		00	00	00	02	uint8_t	type	2	PLAYER_MSGTYPE_CMD
				00	00	00	01	uint8_t	subtype	1	PLAYER_POSITION2D_CMD_VEL
				41	D3	77	40	double	timestamp	1.31E09	time since epoch
				2F	52	D0	9E				
				00	00	00	00	uint8_t	seq	0	transport-specific (unused)
				00	00	00	1C	uint32_t	size	28	length of body (bytes)

2.4.3.3 Player Message Payload. For messages that include commands or data, Player appends a variable-length payload, which is identified in the header. These payloads are also XDR-encoded and defined by the interface they represent (e.g., Position2D). Table 2 shows an example payload for a Position2D command. The first six XDR cells (vx, vy, va) encode the velocity commands as double precision floating point values. The final XDR cell (state) specifies the motor state (on, off).

Table 2. Example Player Command Payload

				4-byte XDR cells				*Type*	*Name*	*Value*	*Description*
<-4*7 = 28-byte body->	player_position2d_cmd_vel	player_pos2d	double	00	00	00	00	double	vx	0.0000	velocity on the X-axis
				00	00	00	00				
			double	00	00	00	00	double	vy	0.0000	velocity on the Y-axis
				00	00	00	00				
			double	BF	F0	C1	52	double	va	-1.0272	angular velocity
				38	2D	73	65				
			int	00	00	00	01	uint8_t	state	1	motor state

2.5 iRobot Create Platform

The iRobot Create is an educational robot platform designed for educators, students, and developers [Iro11]. iRobot provides an interface specification that allows developers to send commands to robot motors and read data from sensors onboard the Create. Player server includes a driver that implements this interface specification. In Figure 9, a Player client connects via WIFI to the Player server that is executed on the Overo Earth embedded system. The server translates the client commands through the Create driver and transmits them via an RS232 serial connection to the Create microcontroller. The Create microcontroller performs the hardware level control and returns requested data to the Player server, which forwards it back to the client.
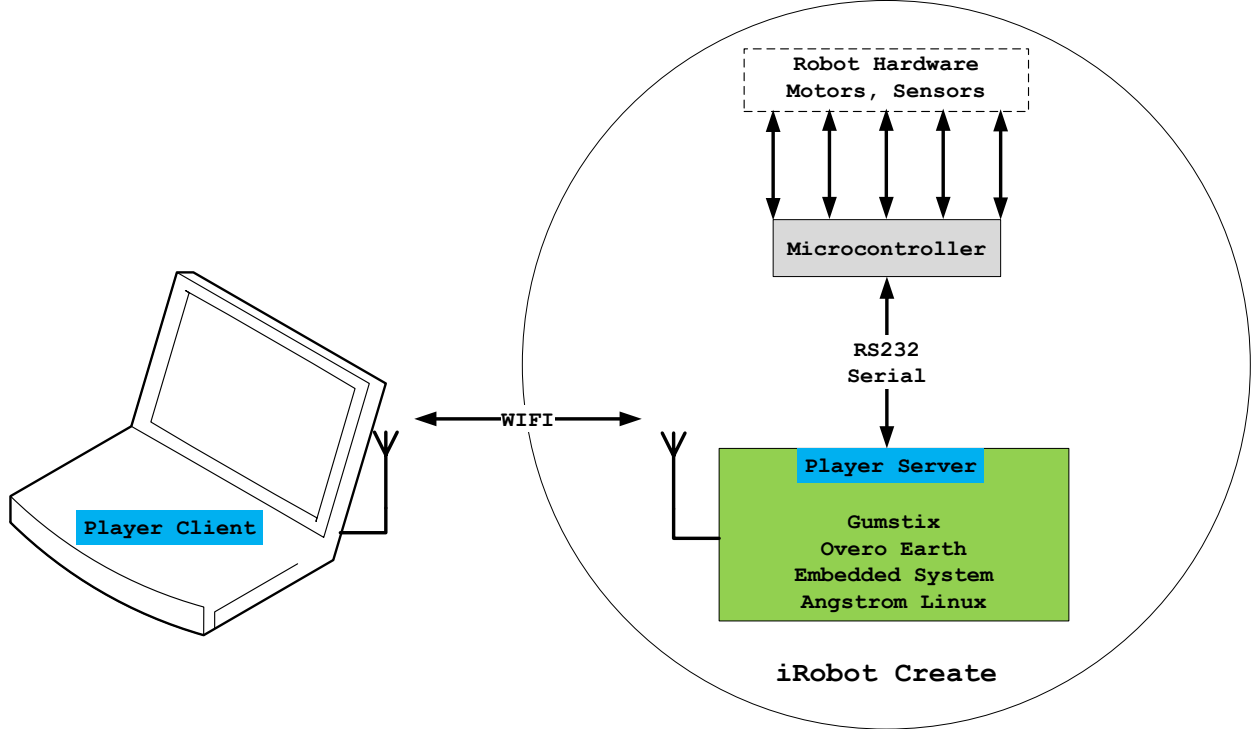
Figure 9. iRobot Create Architecture

2.6 Network Attacks

This section provides an overview of some common network attacks that compromise the CIA security model described in Section 2.2. The naming conventions discussed in Section 2.2 are used in which Alice and Bob are legitimate parties wishing to communicate securely and Mallory is a malicious attacker who tries to compromise their communication. A thorough understanding of these attacks is vital to this research because a subset of the attacks described in this section are implemented in the experiments described in Chapter 3.

2.6.1 Attacks on Confidentiality. Alice and Bob wish to communicate privately over an insecure network. Mallory can read any plaintext sent over the insecure network and thus compromise confidentiality. This type of attack is called *eavesdropping*. To prevent eavesdropping, Alice and Bob can encrypt their communication such that Mallory can only read incomprehensible ciphertext that is transmitted across the network. Encryption techniques fall into two categories: symmetric-key and public-key. In symmetric-key algorithms, Alice and Bob

16

share encryption and decryption keys. Modern examples include the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). In public-key algorithms, Bob's encryption public-key is available to all parties, but the corresponding decryption private-key is known only to Bob. Alice uses Bob's public-key to encrypt her message and only Bob can decrypt it because he is the only one with access to the private-key [TrW06]. RSA is a modern example of a public-key algorithm.

2.6.2 Attacks on Integrity. Alice and Bob wish to communicate while ensuring that they are in fact communicating with each other (authentication) and that their messages have not been altered (data integrity). Mallory can launch a class of attack called *address spoofing,* described by Heberlein and Bishop, in which Mallory uses false IP addresses to establish forged communication with Bob [HeB96].

Another class of integrity compromise is the *man-in-the-middle* (MITM) attack, in which Alice initiates communication with whom she thinks is Bob, but who is actually Mallory. Mallory forwards communication (potentially modified) to and from Alice and Bob who are unaware of Mallory's actions. MITM attacks can be achieved when Mallory is on Alice or Bob's subnet using an attack called *ARP cache poisoning* [Wha01, Phi07]. In this MITM attack, Mallory sends gratuitous ARP responses to remap Alice and Bob's IP addresses to her MAC address. Because of this, Alice and Bob unknowingly send messages meant for each other's MAC addresses to Mallory's MAC address instead.

If Mallory is on the same Ethernet switch as Alice, she can perform another MITM attack called *port stealing* [OrV03]. In this attack, Mallory floods her own Ethernet port with packets containing the MAC address of the gateway router. When Alice sends a message to her gateway router, the switch will incorrectly forward it to Mallory instead because the switch associates the gateway router's MAC address with Mallory's Ethernet port.

TCP connection hijacking is an active MITM attack that exploits TCP sequence numbers to gain control of an ongoing TCP connection [Jon95]. In this attack, Mallory creates a desynchronized state between Alice and Bob's sequence numbers, preventing them from exchanging data directly. Mallory captures the corrupt messages to maliciously modify the application layer and then corrects the sequence numbers so Alice and Bob will accept the modifications. Joncheray proposes two methods for creating the desynchronized state, early desynchronization and null data desynchronization [Jon95]. Early desynchronization is achieved when Mallory resets Alice and Bob's initial TCP connection and then quickly establishes a new malicious connection with one of them. Null data desynchronization involves Mallory watching an ongoing TCP connection and then injecting a large amount of null data with correctly calculated next sequence numbers. One negative side effect of TCP connection hijacking is that it generates an *ACK storm* when the connection becomes flooded with desynchronized ACK packets. The ACK storm can potentially overwhelm the attacker's ability to capture and retransmit packets.

The MITM attacks described above can be mitigated by authenticating the messages Alice and Bob send to one another. *Hash functions* map a large variable-length collection of messages into a small fixed-length set of message digests. These are typically used for error detection. A *cryptographic hash function* provides authentication in addition to integrity through the use of public-key cryptography. Alice first hashes her message to produce a message digest, then encrypts the digest with her private-key to produce a digital signature [BSP95]. *Message authentication codes* (MAC) work similarly to cryptographic hash functions except that rather than using public-key cryptography, Alice's message digest is encrypted with a symmetric-key she shares with Bob [KuR10a].

By capturing and storing messages sent between Alice and Bob, Mallory can launch a third class of integrity attack: the *replay attack*. In this attack, Mallory replays old messages which she might not even be able to decipher because of encryption or modify because of authentication protection [KuR10]. By replaying a message, Mallory could impersonate Alice and compel Bob to repeat some action, resulting in negative consequences. Using a timestamp, nonce (one time use number), or sequence number as input to a cryptographic hash function defeats replay attacks [KuR10a].

2.6.3 Attacks on Availability. Alice and Bob wish to have the ability to communicate when needed. Mallory can launch a class of attacks called *denial-of-service* (DoS) to disrupt Alice and Bob's ability to communicate effectively. Mallory achieves this effect by sending messages to Bob that interfere with his normal operation. Typically this means sending a vast number of messages to overload Bob's resources or the network infrastructure that he uses to communicate. A *distributed-denial-of-service* (DDoS) attack works in the same way, except Mallory coordinates many attacking machines to amplify the resulting damage [MDR05].

SYN flooding is a specific DoS attack in which Mallory creates a large amount of half open TCP connections with Bob [Cer98]. Each time a connection is opened, Bob allocates resources for it. Since Mallory never closes these connections, Bob eventually runs out of resources to allocate for new connections coming from either Mallory or Alice. SYN cookies mitigate SYN flooding by using particular choices of initial TCP sequence numbers and waiting to commit the full amount of resources for a connection until the client has completed the TCP handshake [Ber11].

Mallory exploits the network layer with *Smurf attacks* by sending a broadcast ICMP echo request into a susceptible network with Bob's address as the return address [Cer98]. The Smurf network is used to amplify the effect by using many systems which unwittingly flood Bob

19

with echo replies, disrupting his infrastructure. Chau describes that the first step in defending against Smurf attacks is to prevent one's own network from being a Smurf that is used to attack other networks. Configure the router to block all outbound packets that indicate a source address not contained within the routers internal subnet. In addition, disallow incoming broadcast ICMP packets [Cha04].

TCP connection flooding expands upon the SYN flooding attack by finishing the three-way TCP handshake. Mallory floods Bob with forged TCP SYN packets and listens for Bob's SYN-ACK response. Mallory then completes the connection by replying with an ACK using Bob's SYN-ACK sequence number. A popular version of this attack is the HTTP-GET flood attack [YIS07]. Because the malicious packets have legitimate TCP headers and HTTP payloads, they are difficult to distinguish from legitimate requests and thus more difficult to filter out effectively. Yatagai et al. propose behavioral algorithms to detect and deny malicious requests that complete the TCP handshake [YIS07]. In another mitigation technique, *client puzzles,* a server discerns a client's commitment to making a connection by utilizing some of the client's resources. A puzzle is defined as a task that is difficult to solve by the client but easy to verify by the server. Only after the client returns the solved puzzle will the server allocate resources to the connection. For this strategy to be effective, the client must always commit more resources than the server [ANL01].

TCP reset allows Mallory to close a live connection between Alice and Bob by injecting a spoofed TCP header into their connection with the reset bit set. Watson's results show that practical attacks are possible on the order of 10 seconds when the attacker has the capability to transmit 4370 packets per second [Wat04]. To defeat TCP reset attacks, Watson suggests the use of the optional TCP MD5 header to authenticate each packet and its TCP header which contains the reset flag. Malicious packets that fail the MD5 authentication are silently dropped.

Watson also recommends that the TCP window size be carefully tuned in order to make it as small as possible, while still maintaining reliable and timely communication. A smaller TCP window size forces the attacker to expend more resources to close the connection because the probability that the reset packet correctly falls into the window is reduced [Wat04].

2.6.4 Network Attack Tools. To demonstrate exploits that compromise the Player protocol, several common networking tools are extended by this research. Wireshark is a network protocol analyzer that allows a user to capture and interactively browse computer network traffic. Because it is written in C/C++, it can be compiled on all popular operating systems and supports a large number of protocols out of the box [Wir11]. A custom Wireshark dissector that parses the Player protocol described in Section 2.4.3 is created for this research and included in Appendix A.

Scapy is an interactive packet manipulation program developed as a Python module. The core features it provides are the ability to capture and dissect packets and the ability to forge and transmit packets on the wire. Scapy is designed to handle scanning, trace routing, probing, and network attacks. By including hooks to bind new protocols, Scapy is extendable by the user [Sca11]. Because it is written in Python, Scapy code runs on all popular operating systems and supports popular protocols. The Player protocol described in Section 2.4.3 and the network attacks selected in Section 3.9 are implemented in Scapy.

2.7 Related Works

Caldera et al. extend Network Simulator 2 (NS2) to measure the performance penalties of IPsec and Internet Key Exchange (IKE) on the Mobile IP protocol [CDN00]. The metrics the authors choose are network throughput and delay. Caldera et al. investigate three scenarios involving different combinations of AH and ESP in transport and tunnel modes. Results show that IPsec does not have a significant penalty on throughput relative to the erratic effects introduced by the physical wireless link. Thus for systems communicating wirelessly, Caldera et al. predict that the performance impacts of IPsec are negligible.

Argyroudis et al. investigate the performance impacts of using strong cryptographic protocols (SSL and IPsec) on handheld devices [AVT04]. Their platform consists of an HP iPAQ H3630 with a 206MHz StrongARM processor and 32MB of RAM running Windows CE Pocket PC 2002. Results show that both cryptographic protocols introduce measurable latency but are realistically feasible for securing casual HTTP traffic. However, the authors use significantly slower hardware than is commercially available for similar modern devices, so their conclusions may be outdated.

Elkeelany et al. perform analytical studies to estimate the space and time performance impacts of IPsec AH and ESP when operating with three specific cryptographic algorithms: MD5, SHA-1, and 3DES [EMS02]. In terms of space complexity, results show AH and ESP add an additional 24 and 22 bytes respectively to each IP packet. Their results show that for a 500MIPS machine MD5 can be performed at 340Mbps, SHA-1 at 180Mbps and 3DES at 4Mbps.

2.8 Research Contributions

This research extends work in the area of mobile client-server security. The literature produced by the Player community does not consider the security of the system, an oversight this thesis addresses. This research implements several well-known network attacks against

Player and records the results. Given that robots have clear physical consequences (e.g., safety concerns), it is recommended that the Player community consider how the network protocol could be designed more securely. The performance analysis cited by Argyroudis et al. in Section 2.7 is several years old; this analysis is updated using modern technology here. This research not only quantifies the cost of system resources incurred by IPsec, but also synthesizes this with the security cost associated with an unprotected system. Results are supported with measured experimental data, which complement previous simulated and calculated works cited in Section 2.7 by Caldera et al. and Elkeelany et al.

2.9 Literature Review Summary

This chapter provides the background in network security, security protocols for client-server applications, the Player project, and network attacks necessary to understand this research. The methodology described in Chapter 3 builds upon the works cited in this literature review to accomplish the research goals of this thesis.

III. Methodology

3.1 Chapter Overview

This chapter provides the methodology for analyzing the vulnerability of Player's command and control protocol. The experiments described in this chapter provide data to answer the research questions of this thesis. Analysis of this data is presented in Chapter 4.

Section 3.2 defines the research goals of this thesis. Section 3.3 describes the approach for accomplishing these research goals. Section 3.4 provides the system boundaries that frame the System Under Test (SUT). Section 3.5 lists the services that the SUT provides. Section 3.6 defines the workload that the SUT performs. Section 3.7 defines the metrics by which the performance of the SUT is measured. Section 3.8 defines the system and workload parameters in fine detail so that this work can be replicated. Section 3.9 lists the factors that are expected to affect system performance. Section 3.10 describes the evaluation technique that is used to test the research hypotheses. Section 3.11 provides the experimental design for this work. Finally, Section 3.12 provides a summary for the chapter.

3.2 Research Goals

The research goals of this thesis are threefold:

1) Demonstrate the vulnerability of the Player protocol to network attacks;

2) Demonstrate the effectiveness of IPsec to secure the Player protocol;

3) Quantify the cost of IPsec to secure the Player protocol.

Because the robot is an embedded device, it is resource constrained in both computing capabilities and energy storage. In addition, a mobile robot uses a wireless network to communicate, which is inherently bandwidth-limited. Supposing that IPsec can be shown to be effective at protecting Player, the overhead it introduces could exceed the capabilities of some robot platforms. Therefore, the *cost* for research goal 3 is defined as the additional computing,

energy, and network bandwidth resources a robot running Player must expend to protect its command and control communication with IPsec.

This research also determines the performance impact of running IPsec while under attack. An important question to consider is which types of command and control exploits are mitigated by the different IPsec protocols? What is the performance impact that different IPsec protocols and exploits cause? At a tactical level, this research determines the performance cost of IPsec to defeat particular exploits, as certain IPsec protocols protect different aspects of the CIA security model, described in Section 2.2.

It is hypothesized that exploits against confidentiality, integrity, and availability will be successful in compromising an unprotected Player system. Because IPsec AH authenticates the IP payload, it will defeat attacks against integrity that attempt to inject forged data. IPsec ESP will defeat attacks against confidentiality by encrypting the IP payload, making it infeasible for an attack to read the plaintext. By both authenticating and encrypting the IP payload, IPsec AH+ESP will defeat attacks against both integrity and confidentiality. No IPsec protocol is expected to be effective against availability attacks because IPsec has no mechanism to prevent an attacker from consuming shared network resources. Using IPsec will measurably increase the CPU utilization, energy consumption, and network traffic of the embedded system in the robot because of the extra computational steps needed to perform authentication and encryption algorithms and the addition of the IPsec network headers. Of the two IPsec protocols, AH is expected to consume fewer resources than ESP as Dai finds that with the same key size, SHA-2 consumed fewer cycles per byte than AES [Dai09]. IPsec AH+ESP is expected to consume the most resources because both the AH and ESP protocols are applied.

Attacks targeting confidentiality are not expected to affect CPU utilization, energy consumption, or network load, since they are completely passive. Attacks against integrity and

availability are expected to increase CPU utilization and network load because the attacker will introduce additional packets into the network that will be processed.

3.3 Approach

To accomplish the aforementioned research goals, vulnerabilities in the Player protocol that compromise confidentiality, integrity, or availability are identified. An exploit based on each vulnerability is written to compromise Player communication. A repeatable set of commands a client sends to a robot is defined for each experiment in Section 3.10.3. While the client transmits these baseline commands, the system is subjected to a specific exploit while operating using a specific IPsec protocol. Exploits that compromise confidentiality are successful if they correctly determine the position of the robot. An exploit that successfully injects false position data into the Player connection successfully compromises integrity. Exploits that compromise availability are successful if they terminate the connection between the client and the robot. The performance of the system under these different conditions is measured to quantify the effects of defense protocols and exploits.

3.4 System Boundaries

The System Under Test (SUT) in this research is the Player Defense System (PDS). PDS (Figure 10) consists of Alice (Player client), Bob (Player server), Mallory (malicious attacker), a defense protocol, and the network over which communication occurs. Input to the system is legitimate position commands and position data-requests from the client (Alice) as well as malicious exploits from the attacker (Mallory). Output from the system is both the physical movement of the server (Bob) and position data packets sent in response.

26
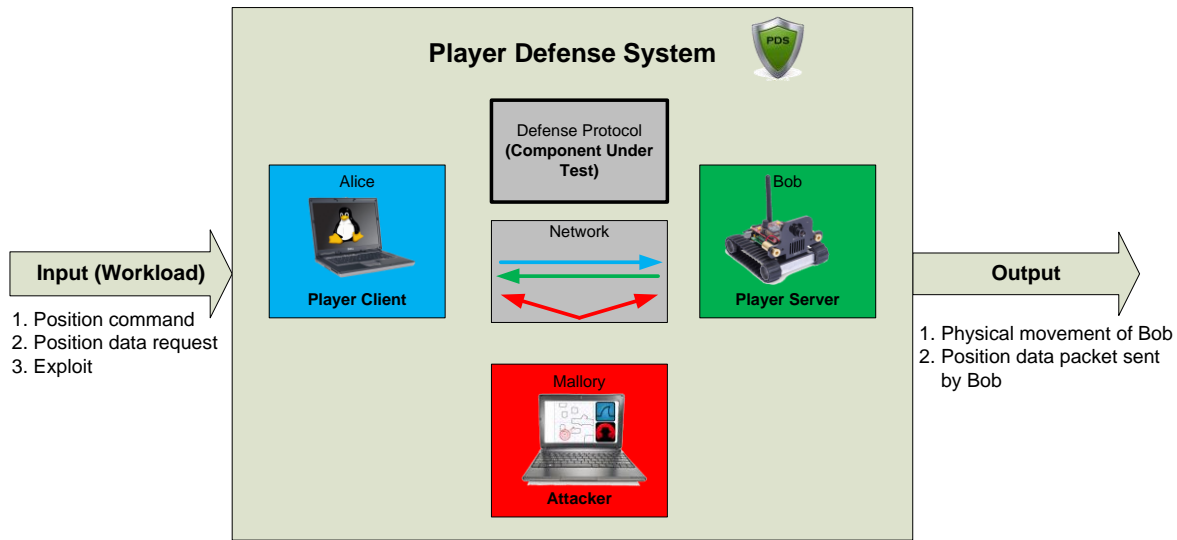
Figure 10. Player Defense System

The following are components of the PDS:

- Alice— Alice is a computer that runs the Player client. She issues position commands over the network to Bob and listens for responses from Bob that either acknowledge her commands or contain Bob's position. Alice must agree with Bob on the defense protocol to properly communicate.

- Bob— Bob is a robot with an embedded system that runs the Player server. Because he is embedded, Bob is more resource constrained than either Alice or Mallory. Bob listens for position commands, acknowledges the commands, and performs hardware actions. If Bob receives a data-request, he replies with a packet containing his current position.

- Mallory— Mallory is a computer that acts as a malicious attacker. Mallory has the ability to intercept network traffic as well as modify and create messages of her own. She launches exploits to violate the confidentiality, integrity, or availability of the command and control communication between Alice and Bob. Mallory has no knowledge of any shared secret between Alice and Bob that may be used in the

27

defense protocol. Mallory does not attempt any brute force attacks to determine authentication or encryption keys.

- Network— The network is a communication channel that supports the coordination of multiple end systems. It provides no confidentiality, integrity, or authentication to the messages sent by end systems. It is a shared medium, meaning systems contend for use of the channel and any messages can be read by all systems.

- Defense Protocol— The defense protocol is the Component Under Test (CUT). It is a security mechanism agreed to by both Alice and Bob in which they use a shared secret to protect one or more principles of the CIA security model.

While Player has been developed to support multiple client and server agents, this research limits the scope to a single client and server to focus on analyzing the vulnerabilities in the Player protocol. Player also supports many different robot services other than 2-dimensional position, but those are also beyond the scope of this research. Exploits employed by Mallory are limited to the following network protocols: Player, TCP, IP, ARP, and Ethernet. That is, Mallory does not attempt to exploit a vulnerability in the Player application itself to execute arbitrary instructions on the end systems of either Alice or Bob with an exploit such as a buffer-overflow attack. Rather, Mallory exploits weaknesses in Player's network protocol to externally cause effects. This distinction is made to focus the scope of this research on Player's network protocol vulnerabilities rather than its software application vulnerabilities.

3.5 System Services

PDS provides two services: a command and control service and defense against exploits. The command and control service receives an input stream of position commands and position data-requests that are transmitted from Alice to Bob. A position command is successful if Bob moves as commanded and fails if he either does not move or moves in a way other than

commanded. A position data-request is successful if Bob replies with a correct position data packet and fails if he either does not reply or replies with incorrect data. Failure modes as described above for the command and control service are not considered part of this research as they do not support the goals defined in Section 3.2.

The second service PDS provides is defense against exploits. Specifically, PDS protects confidentiality, integrity, and availability. The outcome is a success if the system defeats the exploit. The outcome is a failure, however, if the exploit is not defeated. The precise goals of each exploit are enumerated in Section 3.10.3.

3.6 Workload

The workload submitted to PDS is composed of two distinct parts: a stream of commands sent from Alice to Bob to exercise the legitimate command and control of Bob and one of six exploits, launched by Mallory, to demonstrate the defensive service of PDS. Since the goal of this research is to study real-time command and control, the legitimate command stream models a human pilot remotely piloting the robot. This workload is comprised of a continuous, periodic stream of commands. In addition, since a human pilot requires real-time feedback to correctly pilot the robot, the legitimate command stream will also contain a certain ratio of data-requests per commands. These workload parameters are defined in Section 3.8.2.

Exploits are submitted to the system as part of the workload. The six selected exploits are designed to emulate common network attacks that compromise one of the principles of the CIA security model. Each of the selected exploits demonstrates real-world impact on the command and control of a mobile robot and is publically available. These exploits are enumerated in Section 3.8.2.

29

3.7 Performance Metrics

Figure 11. Experimental Parameters, Factors (bold), and Metrics

The following metrics, depicted in Figure 11, measure the performance impact of the defense protocol employed by PDS.

- Exploitation Outcome (Mallory)— Each exploit launched by Mallory has a specific goal to violate one principle of the CIA security model. The success or failure of each exploit is measured based upon whether the PDS defeats or fails to defeat the given exploit. This metric supports research goals 1 and 2 by measuring both vulnerabilities in Player as well as the effectiveness of IPsec.

- CPU Utilization (Bob)— Bob expends CPU execution time (in % utilization) to communicate with Alice, process each legitimate command, and process any

malicious messages from Mallory. Since Bob incurs processing overhead by using the defense protocol, this metric supports research goal 3 by quantifying the CPU utilization overhead of IPsec to secure the Player protocol.

- Power Consumption (Bob)— Bob consumes power (in Watts) to communicate with Alice, process each legitimate command, and process any malicious messages from Mallory. This includes the power consumed by the embedded system including all peripheral IO devices attached to it but excludes the internal iRobot Create microcontroller and hardware, depicted in Figure 9. Because the robot is operating in a mobile environment with limited energy stores, this metric supports research goal 3 by quantifying the energy cost of IPsec to secure the Player protocol.

- Network Load— Alice, Bob, and Mallory transmit messages over the network and consume bandwidth (in Kbps). End systems use a shared, bandwidth-limited network to communicate. This metric supports research goal 3 by quantifying the network bandwidth cost of IPsec to secure the Player protocol.

3.8 Parameters

The system and workload parameters below affect the performance of the PDS.

3.8.1 System Parameters.

- Platform— The platform is the high level computing system that Alice, Bob, and Mallory run on. Configuration instructions for each system are included in Appendix C.

  — Alice— IBM Thinkpad T43 [Len11]

  — Bob— iRobot Create [Iro11], Gumstix Overo Earth [Ove11]

31

— Mallory— Dell Latitude D630 [Del00]

- <u>CPU Type</u>— More powerful CPU architectures at higher clock speeds impact the ability of Alice, Bob, and Mallory to transmit and process messages at higher rates. The following are the CPUs on each system. Bob's CPU directly affects the CPU utilization and power consumption metric.

    — Alice— Intel Pentium M 750 – x86 – 1.86GHz

    — Bob— TI OMAP3530 – ARMv7 – 720MHz

    — Mallory— Intel Core 2 Duo T7500 – x86 – 2.2GHz

- <u>Memory</u>— The following are the amounts of random access memory (RAM) available to Alice, Bob, and Mallory to store temporary information used for computing. A lack of sufficient memory may cause the client or server process to crash.

    — Alice— 1.5GB

    — Bob— 256MB (Pre Sept 2011 Model) [Gum11]

    — Mallory— 2GB

- <u>Network Interface Card (NIC)</u>— The NIC is the peripheral IO device that Alice, Bob, and Mallory use to transmit and receive messages over the network. Each NIC directly affects the network load metric and Bob's NIC also affects power consumption. The NICs used for each system are listed below.

    — Alice— Broadcom NetXtreme Fast Ethernet

    — Bob— SMC 2209USB/ETH

    — Mallory— Broadcom 57XX Gigabit Ethernet

- <u>Operating System (OS)</u>— The OS provides processing and networking services to processes running on Alice, Bob, and Mallory. It affects how network packets are

handled and how kernel routines are executed. Bob's OS directly affects the CPU utilization and power consumption metrics. All systems are configured with the latest stable Linux kernel so that results are applicable to future work.

— Alice— Ubuntu 11.10, Linux Kernel 3.0.0

— Bob— Overo Angstrom omap3-console-image 2011.03, Linux Kernel 3.0.0

— Mallory— Ubuntu 11.10, Linux Kernel 3.0.0

- Network Type— The network type is the physical layer communication channel that Alice, Bob, and Mallory use to communicate. The data-link layer protocol affects the network load metric as it dictates how users encapsulate higher level protocols and handle collisions. The network type used is an IEEE 802.3u 100BASE-TX Ethernet network. All systems share the same collision domain through a HP Procurve 10/100Mbps Hub 12 – J3294A Ethernet hub. Use of a wireless network is excluded from this research because the research goals, defined in Section 3.2, focus on studying vulnerabilities in the Player protocol for a single client and server. A realistic wireless environment would have also introduced additional system parameters. The Ethernet LAN is expected to produce more repeatable results while still providing a shared, bandwidth-limited medium.

- Player Version— Alice and Bob use Player client and server, respectively, to perform command and control services. The version selected affects the CPU utilization and power consumption metrics. Playerjoy is a Player client that is included in the Player project that allows 2-dimentional position control of a server through joystick or keyboard input. The latest stable release is selected so that results are applicable to future work.

— Alice— Playerjoy 3.0.2 (Client)

— Bob— Player 3.0.2 (Server)

33

- Attack Tool— Mallory uses a packet sniffer and injector to dissect and forge packets that implement exploits. The attack tool will affect the network load and the exploitation outcome metrics. The latest release is selected so that results are applicable to future work.

  — Mallory— Scapy 2.2.0, Python 2.7.2

- Defense Protocol— The defense protocol is a security mechanism in which Alice and Bob use some shared secret to protect confidentiality, integrity, or availability. The defense protocols are enumerated in Section 3.9.

- IPsec Implementation— The IPsec implementation consists of the software that implements the IPsec protocol on Alice and Bob. The latest available ipsec-tools package is selected for each operating system distribution listed below. The Angstrom 2011.03 distribution (Bob) has an older version than is available to Ubuntu 11.10 (Alice) but remains functionally the same.

  — Alice— Linux Kernel 3.0.0, RFC 4302 [Ken05] and 4303 [Ken05a] ipsec-tools 0.8.0

  — Bob— Linux Kernel 3.0.0, RFC 4302 [Ken05] and 4303 [Ken05a] ipsec-tools 0.7.2

- Cryptographic Algorithm— IPsec AH uses a hash-based message authentication code (HMAC) to provide authentication. IPsec ESP uses a symmetric-key encryption algorithm to provide confidentiality. The selected cryptographic algorithms affect Bob's CPU and power consumption metrics. These algorithms are selected because they are recommended by NIST [Nis03, Nis08]. Large key sizes are used to make brute force attack infeasible and to induce the highest computational stress an embedded device is likely to encounter.

- — Alice— SHA-256, AES-256

  — Bob— SHA-256, AES-256

- Power Supply— Bob requires power for two separate systems, the iRobot Create robot and the Gumstix Overo Earth embedded system, depicted in Figure 9. A goal of this research is to quantify the cost of protecting the command and control system, so the Overo Earth is powered separately from the Create robot to remove any variation caused by the robot hardware from the power analysis. A detailed wiring diagram of the power subsystem is included in Appendix D.

  — Bob— iRobot Create: Wagan Tech MTR72DAUL-1250A 14VDC 5A Regulated Power Supply

  Gumstix Overo Earth: CUI INC EPS050100 5VDC 1A Regulated Power Supply

3.8.2 Workload Parameters.

- Command Frequency— Alice transmits commands to Bob at a regular interval (in commands/s) to exercise legitimate command and control. Using Playerjoy as the client, a pilot study reveals that the maximum command frequency is limited by the latency of Bob servicing the data-request. Alice waits for Bob to respond with the data before issuing a new command.

  — Alice— 5 commands/s

- Data Request to Command Ratio— The data-request to command ratio is the ratio (in data-requests/command) of data-requests sent from Alice to Bob for every command. This parameter defines the type of feedback a human pilot receives from the robot while remotely piloting it.

  — Alice— 1 data-request/command

- Exploit— Mallory launches one of six specific attacks that each target a single principle of the CIA security model. Each exploit is inseparably related to its exploitation outcome metric and can also affect Bob's CPU utilization, power consumption, and network load. These exploits are enumerated in Section 3.9.

3.9 Factors

Table 3 summarizes the factors and levels used in this research.

Table 3. Factors and Levels

Factor	Levels
Defense Protocol	None, IPsec AH, IPsec ESP, IPsec AH+ESP
Exploit	None, Passive Sniffing, ARP Cache Poisoning, TCP Connection Hijacking, TCP Reset, TCP Connection Flooding

- Defense Protocol— The defense protocol is chosen as a factor because it is the component under test. It is expected to affect all performance metrics. IPsec is used for the defense protocol levels because it provides confidentiality and integrity protection for any application layer protocol without the need to modify that application. Transport mode is selected for this research because the single client-server pair, defined in Section 3.4, resides on the same subnet with no intermediary routers. The IPsec configuration file that defines the security associations used by Alice and Bob is included in Appendix B. PGP, SSL, and SSH are excluded from this research because they require modification to the Player application. Tcpcrypt is also excluded because it is still a work in progress and authentication requires modification to the Player application. The factor levels are

    — None— No defense protocol is used.

    — IPsec Authentication Header (AH)— IPsec with integrity and authentication protection in transport mode.

36

— IPsec Encapsulating Security Protocol (ESP)— IPsec with confidentiality protection and no optional authentication in transport mode.

— IPsec Authentication Header and Encapsulating Security Protocol (AH+ESP)— IPsec AH in transport mode for integrity and authentication of an ESP payload in transport mode for confidentiality protection.

- Exploit— The exploit factor characterizes the ability of the defense protocol to defeat attacks against the CIA security model. Each level below represents a published network attack described in detail in Section 2.5.

  — None— Mallory does not launch an attack against Alice and Bob.

  — Confidentiality: Eavesdropping

    ▪ Passive Sniffing— Mallory passively sniffs and dissects the position data messages Bob transmits to determine the positionX, positionY, and positionA fields.

  — Integrity: Man-in-the-middle (MITM)

    ▪ ARP Cache Poisoning— Mallory sends malicious ARP messages to Alice and Bob to trick them into sending communication to her rather than each other. Mallory then forwards traffic between Alice and Bob and violates the integrity of Bob's position data messages by altering the positionX, positionY, and positionA fields.

    ▪ TCP Connection Hijacking— Mallory sends forged TCP segments to Alice and Bob to desynchronize their connection. Mallory forwards correctly synchronized versions of Alice and Bob's packets and violates the integrity of Bob's data messages by altering the positionX, positionY, and positionA fields.

— Availability: Denial-of-Service (DoS)

- TCP Reset— Mallory sets the TCP reset flag in a forged TCP segment and transmits it to both Alice and Bob to terminate their connection.

- TCP Connection Flooding— Mallory floods Bob with forged TCP SYN packets from pseudo-random IP addresses, listens for Bob's SYN-ACK responses, and sends a forged ACK to finish the spoofed TCP connections. This causes Bob to devote CPU and memory resources to the spoofed connections and eventually causes the Player server to crash.

3.10 Evaluation Technique

3.10.1 Experimental Configuration. To evaluate the system, measurements are taken of an experimental setup of PDS (Figure 12). Measurement is the ideal evaluation technique since many of the metrics are affected by the physical architecture and resources of the end systems. Since the robot is a physical system and moves through its environment, this technique leads to more realistic results that are applicable to other mobile systems.

```
                    ┌─────────────────────────────┐
                    │       Ethernet Hub          │
                    │   192.168.1.XXX Subnet      │
                    │   HP Procurve 10/100Mbps    │
                    │      Hub 12 - J3294A        │
                    └─────────────────────────────┘
```
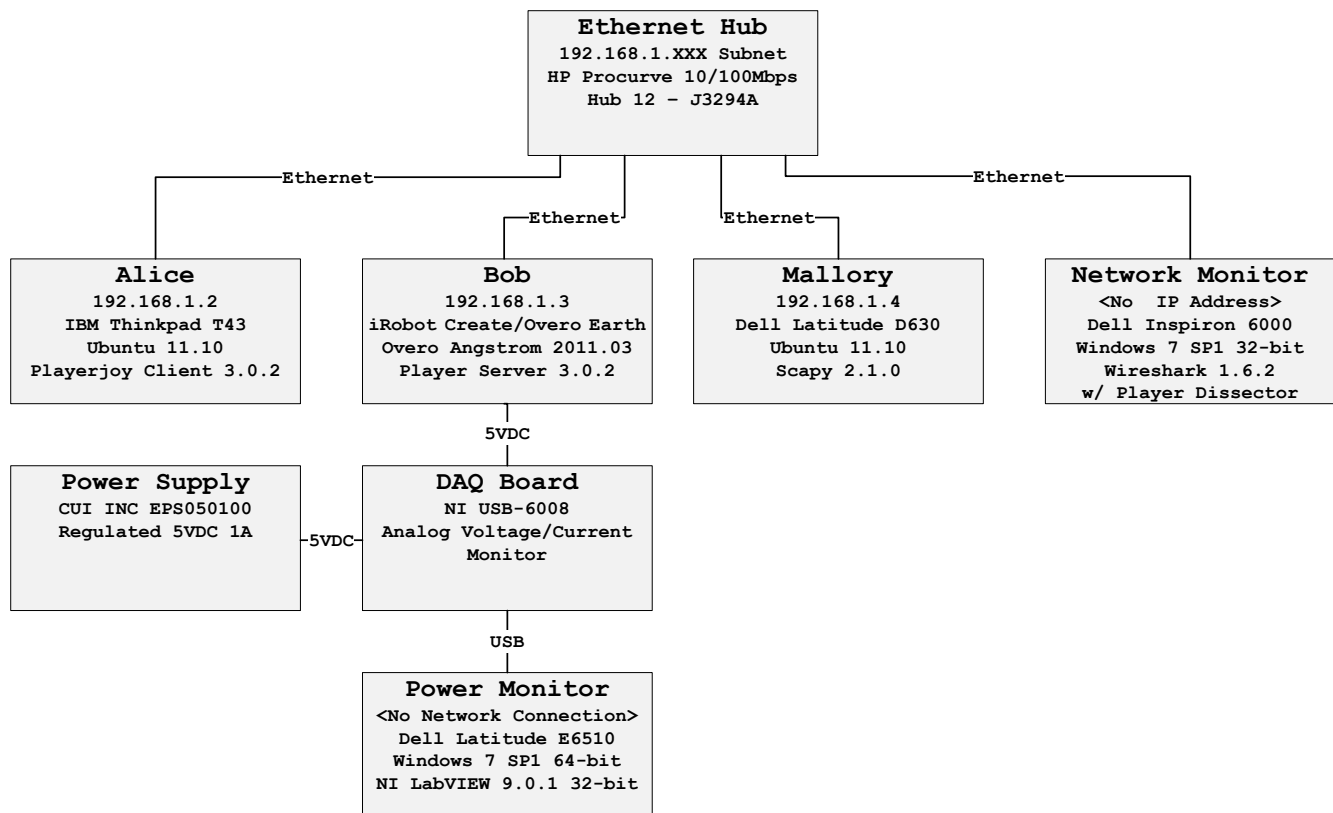
Figure 12. Experimental Configuration

3.10.2 Metric Gathering.

- <u>Exploitation Outcome (Mallory)</u>— Mallory's exploit succeeds if it violates the CIA security model principle that it targets, and fails if it does not. The precise targets for each exploit are listed in Section 3.10.3.

- <u>CPU Utilization (Bob)</u>— The Linux sysstat 9.0.6 package monitors the CPU resources that Bob consumes during an experiment. The command `sar -u 1 80 > cpuUtil.txt` is run on Bob's console to create a tab separated value (TSV) file that contains 80 CPU utilization results each 1 second apart. This corresponds with the experimental timeline provided in Section 3.10.3 in which 80 seconds is selected for the Attack phase. The 1 second interval is selected because it is the highest granularity the sysstat package provides.

39

- Power Consumption (Bob)— The Power Monitor, shown in Figure 12, monitors the power consumption of Bob's embedded system. The Power Monitor consists of a Dell Latitude E6510 running Windows 7 SP1 64-bit and NI LabVIEW 9.0.1 32-bit. It is attached to an NI USB-6008 data acquisition (DAQ) board that samples the voltage and current flowing into Bob's Overo Earth embedded system. Voltage is measured directly using a 12-bit analog to digital converter (ADC), and current is sampled by measuring the voltage drop across a 0.1Ω shunt resistor. A LabVIEW virtual instrument (VI) computes the instantaneous power draw, displays a chart graphing the data over time, and logs the data to a comma separated value (CSV) file. The LabVIEW VI code is included in Appendix E. A detailed wiring diagram of the power monitoring system is available in Appendix D.

- Network Load— The Network Monitor, shown in Figure 12, monitors all network traffic transmitted across the Ethernet Hub. The Network Monitor is a Dell Inspiron 6000 running Windows 7 SP1 32-bit. It uses Wireshark 1.6.2 to passively capture and record all traffic transmitted by Alice, Bob, and Mallory. A custom dissector is added to Wireshark to dissect Player messages into a human readable format. The source code for the Player dissector is available in Appendix A. The total network load (in bytes) is calculated and recorded with the `Statistics > Summary` command in Wireshark.

3.10.3 Experimental Timeline. The timeline used to perform the experiments defined in this chapter is divided into four phases, depicted in Figure 13. In the Setup phase, all of the machines that make up the SUT (Alice, Bob, and Mallory) are powered on and boot into their respective operating systems. Alice and Bob load the appropriate IPsec security associations, as shown in Appendix B, that correspond to the defense protocol factor level being studied. Bob

then launches the Player server, and Alice launches the Player client to connect to the server. During the Setup phase, the Power Monitor and Network Monitor launch LabVIEW and Wireshark, respectively.

After the Player client connects to the server, the Steady State phase is entered. During this phase, the client begins sending velocity commands (vx=0m/s, vy=0m/s, va=0m/s) and position data-requests. Since the robot is never commanded to accelerate, the correct position data from the server will always be: px=0m, py=0m, pa=0m.
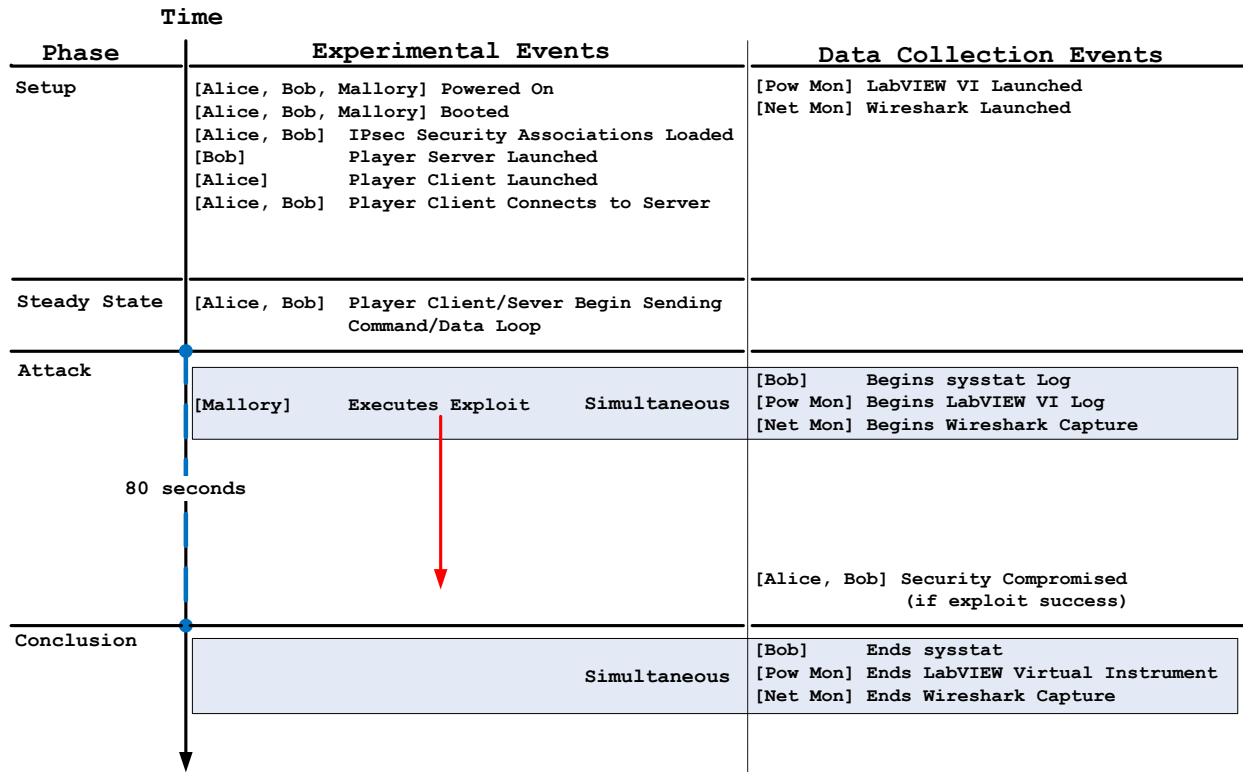
Figure 13. Experimental Timeline

After Steady State has been achieved the Attack phase is executed. Mallory executes the exploit that corresponds to the factor level being studied. Simultaneously, the sysstat log (CPU metric), LabVIEW VI log (power metric), and Wireshark capture (network metric) are started synchronously with an 80 second clock. Eighty seconds is selected as the length of time to collect data because it accommodates the slowest exploit (TCP connection flooding), which

from pilot study data requires 70-75s to successfully overwhelm Bob in this experimental configuration.

After 80 seconds has passed, the Conclusion phase is entered and all data collection is halted. Sysstat, LabVIEW, and Wireshark are configured to halt automatically after the allotted time. Exploit success is determined in three ways based upon which CIA security model principle is targeted. When targeting confidentiality, the exploit is successful if Mallory's console prints out the correct position data (px=0m, py=0m, pa=0m). When targeting integrity, the exploit is successful if Alice's console prints out incorrect position data (anything other than px=0m, py=0m, pa=0m). When targeting availability, the exploit is successful if the connection between Alice and Bob is terminated during the Attack phase.

3.10.4 Validation Strategy. Stage v4.0.1, a Player simulator described in Section 2.4.2, is used to validate Player responses and robot behavior of the PDS under the baseline workload. Performance results are compared with other related works to validate system response [CDN00, EMS02, AVT04]. While the results are not expected to match exactly, there should be a correlation between the results observed in the SUT and those published in the academic literature. Section 4.3.3 validates this research by comparing experimental results with these related works.

3.11 Experimental Design

To measure the relationships between all of the factors listed in Section 3.9, a full factorial design is selected. A total of two factors are chosen with 4 and 6 levels each. A full factorial design requires 4 x 6 = 24 unique experiments. The statistical confidence level is 95%. It is expected that no more than 5 repetitions will be required for a total of 5 x 24 = 120 experiments.

3.12 Methodology Summary

This chapter defined the methodology to 1) Demonstrate the vulnerability of the Player protocol to network attack, 2) Demonstrate the effectiveness of IPsec to secure the Player protocol, and 3) Quantify the cost of IPsec to secure the Player protocol. The Player Defense System (PDS) provides both legitimate command and control services for a Player robot as well as defense against exploits that compromise the confidentiality, integrity, or availability of the system. The components, performance metrics, system and workload parameters, and factors for this research are also defined. The evaluation technique and experimental design are described to allow the experiments herein to be reproduced.

IV. Analysis and Results

4.1 Chapter Overview

This chapter presents an analysis of the data collected from the experiments defined in Chapter 3. Analysis of the data satisfies research goals 1 and 2, defined in Section 3.2, by demonstrating the vulnerabilities of Player and the effectiveness of IPsec. A cost function is defined to satisfy research goal 3 by quantifying the cost of IPsec to secure the Player protocol.

The power consumption metric, detailed in Section 3.7, is excluded from analysis in this chapter for several reasons. Because a wired network is selected, the CPU of the embedded system is the only predominant power consuming device. Since the power consumption of the CPU is directly related to its utilization, power consumption is a redundant metric for this experimental configuration [Tex11]. The power consumption data is none-the-less included in Appendix F to aid future research in this area but is not discussed in this chapter. Chapter 5 provides suggestions for future work related to power analysis.

Section 4.2 describes a cost function to analyze the performance tradeoffs associated with employing IPsec with respect to CPU utilization, network load, and defensive capability. Section 4.3 inputs the data collected in this research into the cost function to determine the cost of employing each defense protocol under a specific scenario. Section 4.4 computes the cost function with respect to two additional scenarios to demonstrate the effect of scenario parameters on cost output. Section 4.5 applies the results from this research to the broader scope of securing Internet client-server communication. Section 4.6 provides a summary of the analysis and results.

4.2 Cost Function

The cost function defined in this section synthesizes three distinct scalar costs (exploit success, CPU utilization, and network load) measured in this research into a single scalar output

(cost). A scenario defines a set of scenario parameters that define how these three measured costs are normalized in the cost function. By returning a single scalar output, results from each defense protocol can be compared to determine the optimal defense protocol for a given scenario.

4.2.1 Cost Function Definition. *Mathematical optimization* is the selection of the best element from some set of alternatives [Dan10]. A subset of the optimization problem is maximizing (utility) or minimizing (cost) a real function. A cost function, therefore, is a real function **f** with an output **f(x)** for which an optimal **x** minimizes **f(x)**.

4.2.2 Cost Function Parameters. Two sets of parameters make up the input of the cost function: scenario and measured parameters. Scenario parameters are selected based on the capabilities of the System Under Test (SUT), knowledge of attacker (Mallory) capabilities, and the value of the secure operation of the system. **{d}** is the set of available defense protocols that the SUT can provide. **{e}** is the set of possible exploits that Mallory can launch against Alice and Bob. Scenario parameters are constant with respect to all combinations of **{d}** and **{e}**. For each exploit **e ∈ {e}**, the probability that Mallory will select exploit **e** is defined as p_e; this parameter weights the likelihood of the possible exploits. The *exploitation outcome probability space* of all p_e naturally sums to 1,

$$\sum_{e \in \{e\}} p_e = 1 \qquad (1)$$

A *cost unit* is defined as the unit by which outputs from the cost function are compared. All variables used in the cost function must be converted to cost units. Cost units are intentionally left ambiguous in the general case and are meant to be replaced with meaningful units when applied to a fielded system. Decision makers could choose alternative units, such as dollars or a metric of mission assurance. The security cost of losing one element of the CIA

security model is defined as s_C, s_I, and s_A in cost units. Because it is assumed for the purposes of this research that each exploit targets only one of the CIA security model principles, s_e is the security cost (s_C, s_I, or s_A) associated with each exploit ($e \in \{e\}$) when it is successful. That is, each exploit can only incur the cost of compromising a single security principle in this particular analytical model. The ratios of CPU utilization and network load to cost units are k_c and k_n, respectively. Scenario parameters are summarized in Table 4. A *scenario* is defined as a particular assignment of these scenario parameters.

Table 4. Scenario Parameters

Parameter	Description	Units
$\{d\}$	Set of available defense protocols	None
$\{e\}$	Set of available exploits	None
p_e	Probability of exploit $e \in \{e\}$ being used	None
s_C	Security cost of losing confidentiality	cost units
s_I	Security cost of losing integrity	cost units
s_A	Security cost of losing availability	cost units
s_e	Security cost (s_C, s_I, or s_A) of exploit $e \in \{e\}$ when successful	cost units
k_c	Ratio of cost units to CPU utilization	$\dfrac{cost\ units}{\%\ utilization}$
k_n	Ratio of cost units to Network load	$\dfrac{cost\ units}{Kbps}$

The measured parameters of the cost function are determined by subjecting the SUT to the available exploits ($e \in \{e\}$) while operating under the available defense protocols ($d \in \{d\}$) and measuring the system response. The CPU utilization is defined as c_e in % utilization. The network load is defined as n_e in Kbps. The probability of exploit success is p_{s_e} and is measured in successes/total attempts. The measured parameters are summarized in Table 5.

Table 5. Measured Parameters

Parameter	Description	Units
c_e	CPU utilization	% utilization
n_e	Network load	Kbps
p_{s_e}	Probability of exploit succeeding	successes/total attempts

4.2.3 Generalized Cost Function. The *generalized cost function* (2) computes the cost of running a defense protocol ($d \in \{d\}$) for the SUT under a specific scenario,

$$f_d = \sum_{e \in \{e\}} p_e \left(k_c c_e + k_n n_e + p_{s_e} s_e \right) \tag{2}$$

Each term is compromised of scenario (Table 4) and measured parameters (Table 5). The summation of all terms represents the total cost for operating a defense protocol under some threat of attack. Output from the cost function is compared to determine which defense protocol results in the lowest cost for a given scenario.

4.2.4 Confidence Interval for Cost Function. The confidence interval of the cost function is calculated using the propagation of uncertainty. Because the terms are linear and uncorrelated, the *propagation of uncertainty for linear, uncorrelated terms* is

$$\sigma_f^2 = \sum_i^n a_i^2 \left(\frac{\sigma_i}{\sqrt{m}} \right)^2 \tag{3}$$

In this equation, n is the number of terms, a_i represents constants of term i, σ_i is the sample standard deviation of sampled values of term i, and m is the number of samples for the measured values. The output, σ_f^2, is the variance of mean, and $\sqrt{\sigma_f^2}$ is the standard error of the mean.

Substituting the terms from the generalized cost function (2) into the propagation of uncertainty (3) creates the *propagation of uncertainty for the generalized cost function,*

$$\sigma_f^2 = \sum_{e \in \{e\}} (p_e k_c)^2 \left(\frac{\sigma_{c_e}}{\sqrt{m}} \right)^2 + (p_e k_n)^2 \left(\frac{\sigma_{n_e}}{\sqrt{m}} \right)^2 + (p_e s_e)^2 \left(\frac{\sigma_{p_{s_e}}}{\sqrt{m}} \right)^2 \tag{4}$$

The *95% confidence interval of the mean* is calculated by inputting the standard error $\left(\sqrt{\sigma_f^2}\right)$ into,

$$f_d \pm 1.96 * \sigma_f \tag{5}$$

4.3 Application of Data to Cost Function

This section covers the parameter selection and data collected from this research for use as input into the cost function defined in Section 4.2. The output of the cost function is compared to determine the defense protocol with the lowest cost.

4.3.1 Scenario Selection. The defense protocol and exploit factor levels chosen in Section 3.9 are used to populate $\{d\}$ and $\{e\}$. Each of the six exploits ($e \in \{e\}$) is assigned equal probability (p_e), $1/6$. This scenario, named the *Likely Exploit Scenario*, weights all exploits equally and presents a high probability of attack ($1- p_{None} = 5/6 = 83.3\%$). The security costs, s_C, s_I, s_A are also equally weighted at 1 cost unit each. The security cost, s_e of each exploit is set to the CIA security principle that it targets with the exception of $s_{None} = 0$.

The maximum measured CPU utilization (c_{max}) and network load (n_{max}) are listed in Table 6. These measured values are used to select the remaining two scenario parameters in the following paragraph.

Table 6. Maximum System Resources

Parameter	*Description*	*Value*	*Units*
c_{max}	Maximum CPU utilization	100	% utilization
n_{max}	Maximum network load	9018	Kbps

The CPU utilization to cost unit ratio (k_c) is set to $1/100$ so that $c_{max} * k_c =$ $1\,cost\,unit = s_a$. The network load to cost unit ratio (k_n) is set to $1/9018$ so that $n_{max} * k_n =$ $1\,cost\,unit = s_a$. This selection of k_c and k_n equates consuming all CPU or network resources

with the cost of losing availability. The PDS scenario parameters for the Likely Exploit

Scenario are summarized in Table 7.

Table 7. Scenario Parameters: Likely Exploit Scenario

Parameter	Value	Units
{d}	{None, IPsec AH, IPsec ESP, IPsec AH+ESP}	None
{e}	{None, Passive Sniffing, ARP Cache Poisoning, TCP Connection Hijacking, TCP Reset, TCP Connection Flooding}	None
p_{None}	1/6	None
p_{Eaves}	1/6	None
p_{ARP}	1/6	None
p_{Hijack}	1/6	None
p_{Reset}	1/6	None
p_{Flood}	1/6	None
s_C	1	cost units
s_I	1	cost units
s_A	1	cost units
s_{None}	0	cost units
s_{Eaves}	s_C	cost units
s_{ARP}	s_I	cost units
s_{Hijack}	s_I	cost units
s_{Reset}	s_A	cost units
s_{Flood}	s_A	cost units
k_c	1/100	$\dfrac{cost\ units}{\%\ utilization}$
k_n	1/9018	$\dfrac{cost\ units}{Kbps}$

4.3.2 Measured Parameters. The measured parameters are populated from the

experimental data collected from the experiments described in Chapter 3. The measured

parameter tables in this section (Table 8, 9, and 10) list the average values for the five

replications of each factor level combination. The raw data for each factor level combination is

included in Appendix F.

Table 8 shows the measured results of the probability of exploit success (p_{s_e}) against the

available defense protocols in successes/total attempts. This data satisfies research goal 1,

defined in Section 3.2, by demonstrating that the Player protocol is vulnerable to attack. Note

that every exploit succeeds when no defense protocol is used. This is because neither the TCP header nor Player payload are authenticated or encrypted. TCP connection hijacking has a success rate of only 80% because of the large amount of traffic (ACK storm) it generates during the attack. It is hypothesized that Alice occasionally does not correctly receive forged Player data either because Mallory's exploit is overwhelmed and becomes unresponsive, or the excessive ACK storm packets collide with the forged packets on the Ethernet hub.

The exploitation success data (Table 8) also satisfies research goal 2 by demonstrating that IPsec is able to protect the Player protocol. Passive sniffing is successful against IPsec AH because Mallory is able to dissect the unencrypted Player payload. IPsec AH defeats all other exploits because it authenticates the TCP header and Player payload. IPsec ESP defeats all of the tested exploits because Scapy is unable to dissect the encrypted TCP header and Player payload. However, it should be noted that it is conceivable that attacks against IPsec ESP in encrypt-only mode, similar to those Paterson and Yau demonstrate, could be used against Player [PaY06]. For example, toggling the TCP reset flag of encrypted Player packets could lead to a successful TCP reset attack. So while IPsec ESP defeats integrity exploits described in Section 3.9, it does not ensure integrity protection in encrypt-only mode. IPsec AH+ESP defeats all exploits described in Section 3.9 because it both authenticates and encrypts the transport and application layers. The attacks implemented in this research are infeasible against IPsec AH+ESP.

Table 8. Measured Probability of Exploit Success (successes/total attempts)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	NA	NA	NA	NA
Passive Sniffing	5/5	5/5	0/5	0/5
ARP Cache Poisoning	5/5	0/5	0/5	0/5
TCP Connection Hijacking	4/5	0/5	0/5	0/5
TCP Reset	5/5	0/5	0/5	0/5
TCP Connection Flooding	5/5	0/5	0/5	0/5

The measured CPU utilization (c_e) in % utilization is given in Table 9. The amount of CPU overhead introduced solely by IPsec is determined by examining the row where no exploit is used (None). CPU utilization increases as expected when the AH and ESP protocols are applied due to the computational load of SHA-256 and AES-256, respectively. IPsec AH+ESP results in an increase of 0.52% CPU utilization (16% increase relative to no defense protocol). The SUT computational resources can easily handle protecting Player with IPsec AH+ESP. When no defense protocol is used, TCP connection hijacking results in 95.66% CPU utilization because it generates an ACK Storm, described in Section 2.5.2. TCP connection flooding results in 40.16% CPU utilization because Bob allocates resources for numerous spoofed TCP connections during this attack. TCP reset has the opposite effect and lowers the CPU utilization by 0.35% utilization because it terminates the TCP connection between Alice and Bob.

Table 9. Measured CPU Utilization (% utilization)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	3.26	3.40	3.41	3.78
Passive Sniffing	3.26	3.40	3.41	3.78
ARP Cache Poisoning	3.79	3.10	4.54	4.06
TCP Connection Hijacking	95.66	3.50	3.22	4.05
TCP Reset	2.91	3.33	3.52	3.86
TCP Connection Flooding	40.16	3.56	3.97	4.32

The measured network load (n_e) in Kbps is given in Table 10. The amount of network overhead introduced solely by IPsec is determined by examining the row where no exploit is used (None). Network load increases as expected due to the extra headers added by IPsec AH and ESP. IPsec AH+ESP increases network load by 22.9Kbps (64.3% increase relative to no defense protocol). The SUT's network resources can easily handle a single Player client-server pair protected by IPsec AH+ESP, as 58.5Kbps accounts for only 0.6% of the maximum measured network bandwidth (n_{max}) defined in Table 6. When no defense protocol is used, ARP cache poisoning doubles the consumed network resources because every message is forwarded again into the network by Mallory with a corrected MAC address. TCP connection hijacking creates a desynchronized TCP state between Alice and Bob, which generates an ACK storm and consumes all of the available network bandwidth, 9018Kbps (n_{max}). Because TCP reset quickly terminates the connection between Alice and Bob, no new network traffic is transmitted after the attack is successful. TCP connection flooding increases the network load by 68.4Kbps due to Mallory's spoofed SYN packets that she sends to Bob, Bob's SYN-ACK responses, and Mallory's subsequent spoofed ACK's.

Table 10. Measured Network Load (Kbps)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	35.6	44.8	49.3	58.5
Passive Sniffing	35.6	44.8	49.3	58.5
ARP Cache Poisoning	70.3	0.5	98.6	116.5
TCP Connection Hijacking	9018	80.5	49.3	58.4
TCP Reset	0.1	79.3	49.2	58.2
TCP Connection Flooding	104.0	84.1	90.8	97.5

4.3.3 Validation. Results published by Elkeelany et al. validate the network load metrics gathered for this research [EMS02]. Their work shows IPsec AH increases the packet size by 24 bytes due to 12 bytes of fixed header and 12 bytes of Integrity Check Value (ICV). Table 11

shows the average measured packet size for the data collected in this research. These values are calculated by dividing the total network load in bytes by the number of packets transmitted (Appendix F). IPsec AH increases the measured packet size by the expected 24 bytes. Elkeelany et al. also conclude that IPsec ESP increases the packet size by 22 bytes due to 10 bytes of fixed header and 12 bytes of ICV. The authors note that ESP may use additional padding (up to 255 bytes), depending on the selected encryption algorithm. Table 11 shows the average measured packet size (in bytes) for this research increased by 36 bytes as a result of ESP. Note that ESP is not configured to use optional authentication (i.e., no ICV is appended) in these experiments, so the expected overhead of ESP is 10 bytes of fixed header plus encryption padding. In this research the encryption algorithm, AES-256, is responsible for 26 bytes of additional padding on average. The IPsec AH+ESP network load reported in Section 4.3.2 is validated because its overhead is the expected sum of both AH and ESP: 24 bytes + 36 bytes = 60 bytes.

Table 11. Average Measured Packet Size (bytes)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	94	118	130	154
Overhead	0	24	36	60

Results from Argyroudis et al. validate the CPU metrics gathered in this research [AVT04]. The authors find that the network security protocols, SSL and IPsec, do not significantly impact real-time communication on mobile devices. An older HP iPAQ H3630 with a 206MHz StrongARM processor and 32MB RAM is tested in their research. Their system has significantly less resources than the embedded system (Bob) specified in Section 3.8. As a result, it is expected that the SUT's CPU resources in this research are affected even less by IPsec. The measured CPU utilization (Table 9) confirms that the CPU overhead is relatively

little (0.52% utilization) when using the most comprehensive protection, IPsec AH+ESP, thus validating this data.

4.3.4 PDS Cost Function Results. The generalized cost function (2) is expanded with the exploits ({e}) and security costs (s_e) defined for the selected scenario parameters (Table 7) to compose the *expanded cost function,*

$$
\begin{aligned}
f_d = p_{None}(k_c c_{None} &+ k_n n_{None} + p_{s_{None}} 0) \\
&+ p_{Sniff}(k_c c_{Sniff} + k_n n_{Sniff} + p_{s_{Sniff}} s_C) \\
&+ p_{ARP}(k_c c_{ARP} + k_n n_{ARP} + p_{s_{ARP}} s_I) \\
&+ p_{Hijack}\left(k_c c_{Hijack} + k_n n_{Hijack} + p_{s_{Hijack}} s_I\right) \\
&+ p_{Reset}(k_c c_{Reset} + k_n n_{Reset} + p_{s_{Reset}} s_A) \\
&+ p_{Flood}(k_c c_{Flood} + k_n n_{Flood} + p_{s_{Flood}} s_A)
\end{aligned}
\tag{6}
$$

Results from the expanded cost function using as inputs the selected scenario parameters (Table 7) and the measured parameters (Tables 8, 9, and 10) are totaled for each term in Table 12. Comparison between the total costs of each defense protocol is reserved until the confidence intervals are calculated later in this section. When no defense protocol is used, the system incurs large cost from each exploit because each exploit successfully compromises the security cost (s_e) associated with it. IPsec AH in penalized by one security cost because it does not defeat passive sniffing. Most notable in Table 12 is the contrast between cells in which exploits are successful (>0.17 cost units) and the cells in which exploits are not successful (<0.17 cost units). This is a result of both the scenario parameter selection in Table 7 and the low CPU and network overhead IPsec introduces relative to the maximum resources available to the SUT.

Table 12. Cost Function Results: Likely Exploit Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	0.0061	0.0065	0.0066	0.0074
Passive Sniffing	0.1728	0.1731	0.0066	0.0074
ARP Cache Poisoning	0.1743	0.0052	0.0094	0.0089
TCP Connection Hijacking	0.4583	0.0073	0.0063	0.0078
TCP Reset	0.1715	0.0070	0.0068	0.0075
TCP Connection Flooding	0.2355	0.0075	0.0083	0.0090
Total Cost, f_d	**1.2184**	**0.2066**	**0.0439**	**0.0480**

The *expanded propagation of uncertainty function* (7) is composed from the propagation of uncertainty for the generalized cost function (4) and the exploits ({e}) defined for the scenario parameters in Table 7. The number of replications, m, for this research is 5.

$$\sigma_f^2 = (p_{None}k_c)^2 \left(\frac{\sigma_{c_{None}}}{\sqrt{m}}\right)^2 + (p_{None}k_n)^2 \left(\frac{\sigma_{n_{None}}}{\sqrt{m}}\right)^2 + (p_{None}s_{None})^2 \left(\frac{\sigma_{p_{sNone}}}{\sqrt{m}}\right)^2$$

$$+ (p_{Sniff}k_c)^2 \left(\frac{\sigma_{c_{Sniff}}}{\sqrt{m}}\right)^2 + (p_{Sniff}k_n)^2 \left(\frac{\sigma_{n_{Sniff}}}{\sqrt{m}}\right)^2 + (p_{Sniff}s_{Sniff})^2 \left(\frac{\sigma_{p_{sSniff}}}{\sqrt{m}}\right)^2$$

$$+ (p_{ARP}k_c)^2 \left(\frac{\sigma_{c_{ARP}}}{\sqrt{m}}\right)^2 + (p_{ARP}k_n)^2 \left(\frac{\sigma_{n_{ARP}}}{\sqrt{m}}\right)^2 + (p_{ARP}s_{ARP})^2 \left(\frac{\sigma_{p_{sARP}}}{\sqrt{m}}\right)^2 \qquad (7)$$

$$+ (p_{Hijack}k_c)^2 \left(\frac{\sigma_{c_{Hijack}}}{\sqrt{m}}\right)^2 + (p_{Hijack}k_n)^2 \left(\frac{\sigma_{n_{Hijack}}}{\sqrt{m}}\right)^2 + (p_{Hijack}s_{Hijack})^2 \left(\frac{\sigma_{p_{sHijack}}}{\sqrt{m}}\right)$$

$$+ (p_{Flood}k_c)^2 \left(\frac{\sigma_{c_{Flood}}}{\sqrt{m}}\right)^2 + (p_{Flood}k_n)^2 \left(\frac{\sigma_{n_{Flood}}}{\sqrt{m}}\right)^2 + (p_{Flood}s_{Flood})^2 \left(\frac{\sigma_{p_{sFlood}}}{\sqrt{m}}\right)^2$$

The standard error of the mean of the expanded cost function (Table 13) is calculated by inputting the scenario parameters (Table 7) and measured parameters (Tables 8, 9, and 10) into the expanded propagation of uncertainty cost function (7). TCP connection hijacking introduces a notably large amount of error from its probability of exploit success (p_{s_e}) term, which propagates to create a large total standard error when no defense protocol is used (None).

This could be mitigated by running additional replications, but as shown later in this section, the resulting confidence interval is sufficiently distinct to draw conclusions without them.

Table 13. Standard Error: Likely Exploit Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	4.851E-07	8.457E-07	2.23E-06	3.597E-07
Passive Sniffing	1.347E-08	2.349E-08	6.19E-08	9.993E-09
ARP Cache Poisoning	3.332E-07	1.440E-07	5.13E-07	1.457E-08
TCP Connection Hijacking	0.001112	1.764E-08	1.39E-08	8.692E-09
TCP Reset	2.234E-08	9.453E-10	8.92E-08	3.950E-08
TCP Connection Flooding	4.931E-07	9.796E-09	1.47E-07	1.539E-07
Total Variance of the mean, σ_f^2	0.0011136	1.041E-06	3.05E-06	5.864E-07
Total Standard Error of the mean, σ_f	*0.03337*	*0.00102*	*0.00175*	*0.00077*

The 95% confidence interval for each defense protocol cost (Table 14) is calculated by populating the 95% confidence interval equation for the mean (5) with the cost function results (Table 12) and the standard error (Table 13).

Table 14. 95% Confidence Interval: Likely Exploit Scenario (cost units)

	None	IPsec AH	IPsec ESP	IPsec AH+ESP
Lower 95% Limit	1.1530	0.2046	0.0404	0.0465
Mean	1.2184	0.2066	0.0439	0.0480
Upper 95% Limit	1.2838	0.2086	0.0473	0.0495

Figure 14 plots the 95% confidence interval of the mean of the cost function results (Table 14). The results show that IPsec ESP and IPsec AH+ESP result in the lowest cost. These minimum costs are a consequence of both the ability to defeat all of the exploits in this research, as well as the relatively low overhead IPsec introduces with respect to the total CPU and network resources of the SUT. Since the confidence intervals for IPsec ESP and IPsec AH+ESP overlap, this experiment cannot differentiate them for this scenario. It is hypothesized that with additional tests, IPsec ESP would result in a statistically-significant

lower cost because IPsec ESP consumes fewer CPU and network resources than IPsec AH+ESP, while still defeating all of the exploits. It is important to note that Patterson and Yau demonstrate attacks on IPsec ESP when not used in conjunction with authentication [PaY06]. Thus, for the Likely Exploit Scenario, IPsec AH+ESP would be the preferred defense protocol given its small additional cost over IPsec ESP and the security benefits authentication provides.
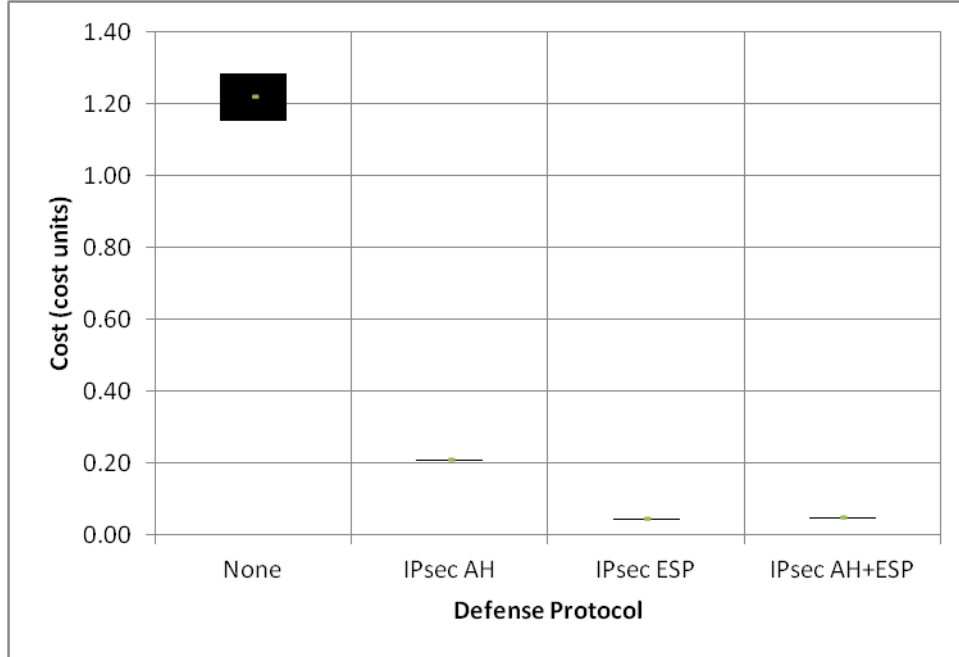
Figure 14. 95% Confidence Interval: Likely Exploit Scenario

4.4 Scenario Exploration

Section 4.3 applies the data gathered from this research to the cost function developed in Section 4.2 using a particular set of scenario parameters defined as the Likely Exploit Scenario. This section demonstrates the importance of selecting appropriate scenario parameters by computing the cost function using the same measured data but with different scenario parameters. The cost results for each additional scenario are compared with the Likely Exploit Scenario.

4.4.1 Confidentiality Free Scenario. The *Confidentiality Free Scenario* is a small deviation from the Likely Exploit Scenario in which the cost of losing confidentiality (s_C) is set

57

to 0. This represents a system in which there are no negative consequences if Mallory reads information as long as she cannot modify it or inject false information. The remaining scenario parameters are left unchanged and are specified in Table 15.

Table 15. Scenario Parameters: Confidentiality Free Scenario

Parameter	Value	Units
{d}	{None, IPsec AH, IPsec ESP, IPsec AH+ESP}	None
{e}	{None, Passive Sniffing, ARP Cache Poisoning, TCP Connection Hijacking, TCP Reset, TCP Connection Flooding}	None
p_{None}	1/6	None
p_{Eaves}	1/6	None
p_{ARP}	1/6	None
p_{Hijack}	1/6	None
p_{Reset}	1/6	None
p_{Flood}	1/6	None
s_C	0	cost units
s_I	1	cost units
s_A	1	cost units
s_{None}	0	cost units
s_{Eaves}	s_C	cost units
s_{ARP}	s_I	cost units
s_{Hijack}	s_I	cost units
s_{Reset}	s_A	cost units
s_{Flood}	s_A	cost units
k_c	1/100	$\dfrac{cost\ units}{\%\ utilization}$
k_n	1/9018	$\dfrac{cost\ units}{Kbps}$

Using the scenario parameters for the Confidentiality Free Scenario (Table 15) and the measured parameters (Tables 8, 9, and 10), the expanded cost function (6) is recomputed into Table 16.

Table 16. Cost Function Results: Confidentiality Free Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	0.0061	0.0065	0.0066	0.0074
Passive Sniffing	0.0061	0.0065	0.0066	0.0074
ARP Cache Poisoning	0.1743	0.0052	0.0094	0.0089
TCP Connection Hijacking	0.4583	0.0073	0.0063	0.0078
TCP Reset	0.1715	0.0070	0.0068	0.0075
TCP Connection Flooding	0.2355	0.0075	0.0083	0.0090
Total Cost	*1.0518*	*0.0399*	*0.0439*	*0.0480*

The standard error of the mean of the expanded cost function (Table 17) is calculated by inputting the scenario parameters for the Confidentiality Free Scenario (Table 15) and the measured parameters (Tables 8, 9, and 10) into the expanded propagation of uncertainty function (7).

Table 17. Standard Error: Confidentiality Free Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	4.851E-07	8.457E-07	2.23E-06	3.597E-07
Passive Sniffing	1.347E-08	2.349E-08	6.19E-08	9.993E-09
ARP Cache Poisoning	3.332E-07	1.440E-07	5.13E-07	1.457E-08
TCP Connection Hijacking	0.001112307	1.764E-08	1.39E-08	8.692E-09
TCP Reset	2.234E-08	9.453E-10	8.92E-08	3.950E-08
TCP Connection Flooding	4.931E-07	9.796E-09	1.47E-07	1.539E-07
Total Variance of the mean, σ_f^2	0.0011136	1.041E-06	3.05E-06	5.864E-07
Total Standard Error of the mean, σ_f	*0.03337*	*0.00102*	*0.00175*	*0.00077*

The 95% confidence interval for each defense protocol cost is calculated in Table 18 by populating the 95% confidence interval equation for the mean (5) with the cost function results (Table 16) and standard error (Table 17) for the Confidentiality Free Scenario.

59

Table 18. 95% Confidence Interval: Confidentiality Free Scenario (cost units)

	None	*IPsec AH*	*IPsec ESP*	*IPsec AH+ESP*
Lower 95% Limit	0.9864	0.0379	0.0404	0.0465
Mean	1.0518	0.0399	0.0439	0.0480
Upper 95% Limit	1.1172	0.0419	0.0473	0.0495

Figure 15 plots the 95% confidence intervals from Table 18 for all cases except when no defense protocol is used. The case when no defense protocol is used is omitted because of its relatively large value and to highlight the relationships between remaining defense protocols. The stark difference between the Confidentiality Free Scenario and the Likely Exploit Scenario, examined in Section 4.3, is that IPsec AH, rather than IPsec AH+ESP, results in the lowest cost. This is expected since the only attack that is successful against it, passive sniffing, imposes a security cost of confidentiality (s_C), which is set to 0 in this scenario. For the Confidentiality Free Scenario, IPsec AH is the preferred protocol due to its low overhead and ability to defeat the consequential exploits.
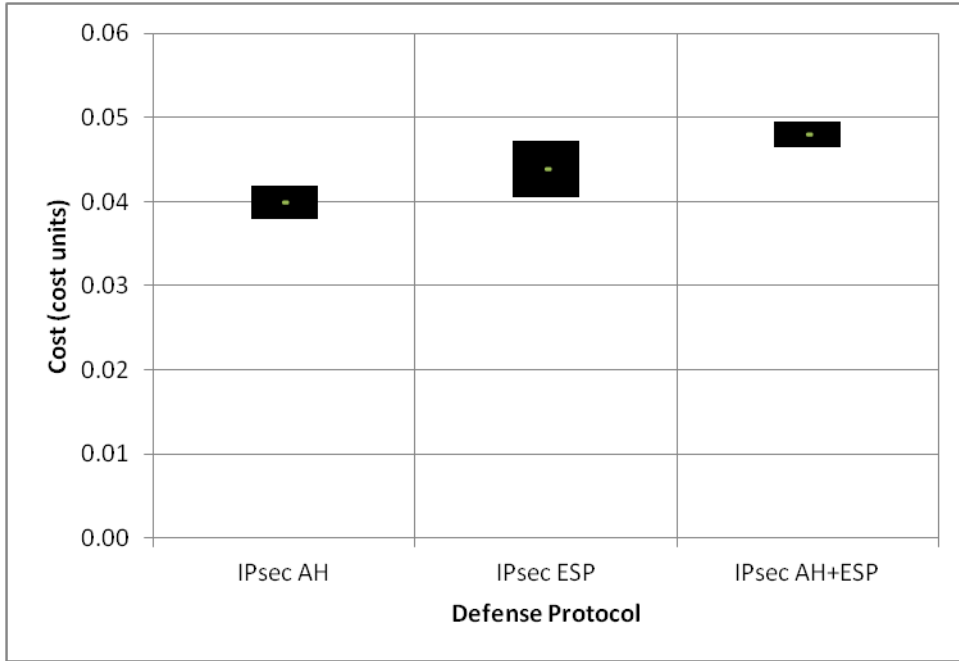
Figure 15. 95% Confidence Interval: Confidentiality Free Scenario
(None defense protocol omitted)

4.4.2 Unlikely Exploit Scenario. The *Unlikely Exploit Scenario* differs from the Likely Exploit Scenario by raising the likelihood that no exploit is used. This represents the scenario in which it is very unlikely that Mallory will attempt to launch exploits. The probably that no exploit is launched (p_{None}) is set to 0.999 and all other exploit probabilities are set to 0.001/5. All other scenario parameters remain the same as the Likely Exploit Scenario, as shown in Table 19.

Table 19. Scenario Parameters: Unlikely Exploit Scenario

Parameter	*Value*	*Units*
{d}	{None, IPsec AH, IPsec ESP, IPsec AH+ESP}	None
{e}	{None, Passive Sniffing, ARP Cache Poisoning, TCP Connection Hijacking, TCP Reset, TCP Connection Flooding}	None
p_{None}	0.999	None
p_{Eaves}	0.001/5	None
p_{ARP}	0.001/5	None
p_{Hijack}	0.001/5	None
p_{Reset}	0.001/5	None
p_{Flood}	0.001/5	None
s_C	1	cost units
s_I	1	cost units
s_A	1	cost units
s_{None}	0	cost units
s_{Eaves}	s_C	cost units
s_{ARP}	s_I	cost units
s_{Hijack}	s_I	cost units
s_{Reset}	s_A	cost units
s_{Flood}	s_A	cost units
k_c	1/100	$\dfrac{cost\ units}{\%\ utilization}$
k_n	1/9018	$\dfrac{cost\ units}{Kbps}$

Using the scenario parameters for the Unlikely Exploit Scenario (Table 19) and the measured parameters (Tables 8, 9, and 10), the expanded cost function (6) is recomputed into Table 20.

Table 20. Cost Function Results: Unlikely Exploit Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	0.0061	0.0065	0.0066	0.0074
Passive Sniffing	0.0061	0.0065	0.0066	0.0074
ARP Cache Poisoning	0.1743	0.0052	0.0094	0.0089
TCP Connection Hijacking	0.4583	0.0073	0.0063	0.0078
TCP Reset	0.1715	0.0070	0.0068	0.0075
TCP Connection Flooding	0.2355	0.0075	0.0083	0.0090
Total Cost	*1.0518*	*0.0399*	*0.0439*	*0.0480*

The standard error of the mean of the expanded cost function (Table 21) is calculated by inputting the scenario parameters for the Confidentiality Free Scenario (Table 19) and the measured parameters (Tables 8, 9, and 10) into the expanded propagation of uncertainty function (7).

Table 21. Standard Error: Unlikely Exploit Scenario (cost units)

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	4.851E-07	8.457E-07	2.23E-06	3.597E-07
Passive Sniffing	4.842E-07	8.440E-07	2.22E-06	3.590E-07
ARP Cache Poisoning	4.798E-13	2.074E-13	7.38E-13	2.098E-14
TCP Connection Hijacking	1.601E-09	2.541E-14	2.01E-14	1.251E-14
TCP Reset	3.217E-14	1.361E-15	1.28E-13	5.688E-14
TCP Connection Flooding	7.101E-13	1.410E-14	2.12E-13	2.216E-13
Total Variance of the mean, σ_f^2	9.710E-07	1.689E-06	4.45E-06	7.187E-07
Total Standard Error of the mean, σ_f	*0.00099*	*0.00130*	*0.00211*	*0.00085*

The 95% confidence interval for each defense protocol cost is calculated in Table 22 by populating the 95% confidence interval equation for the mean (5) with the cost function results (Table 20) and standard error (Table 21) for the Confidentiality Free Scenario.

Table 22. 95% Confidence Interval: Unlikely Exploit Scenario (cost units)

	None	*IPsec AH*	*IPsec ESP*	*IPsec AH+ESP*
Lower 95% Limit	0.0361	0.0365	0.0354	0.0426
Mean	0.0380	0.0391	0.0396	0.0443
Upper 95% Limit	0.0399	0.0416	0.0437	0.0459

Figure 16 plots the 95% confidence interval of the mean for the cost function results (Table 22) against each defense protocol. Compared with the results for the Likely Exploit Scenario, using no defense protocol results in a lower cost than IPsec AH+ESP. This is a result of the probability of no exploit being set so high ($p_{None} = 0.999$) that the CPU and network load overheads begin to dominate the cost function output rather than the associated security costs. Similar costs values result when the security costs (s_C, s_I, and s_A) are reduced to extremely low levels. This highlights possible scenarios where adding authentication and encryption is not ideal due the low probability of attack or low value of the information or service the system provides.
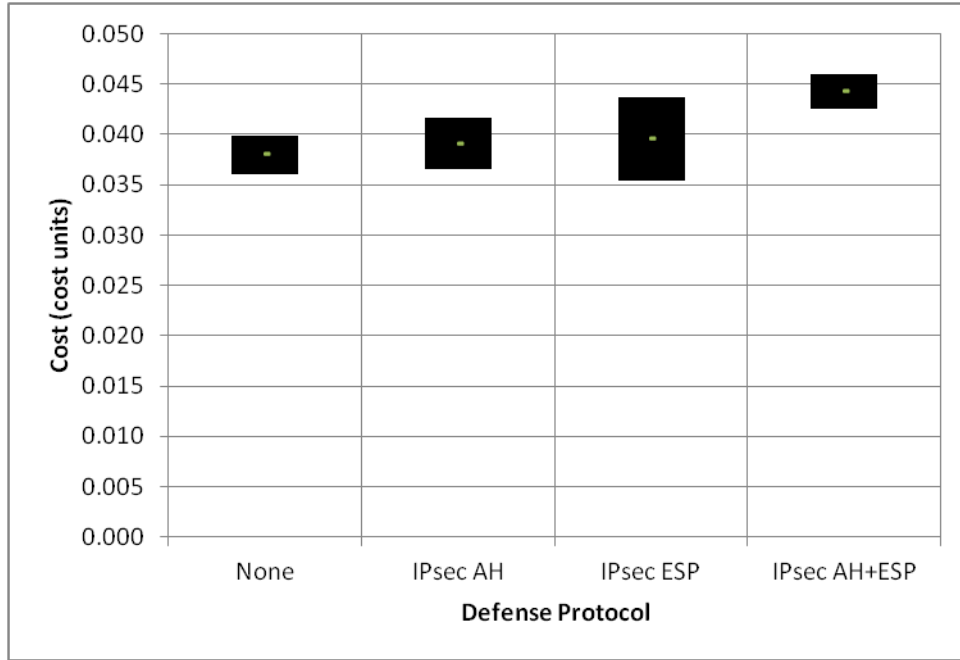
Figure 16. 95% Confidence Interval: Unlikely Exploit Scenario

4.5 Application of Results

This section frames the results of this research within the wider context of network security. Results support the case for authentication and encryption by default due to modern hardware advances. Additionally, this research highlights the trend that as a Player system is scaled to include additional hosts, it is expected that the network overhead introduced by IPsec will become more consequential than the CPU overhead. Finally, results from this research highlights potential weaknesses in the ability of application layer security protocols (e.g., SSL) to protect against DoS attacks that target the TCP connection and supports the adoption of transport layer protocols (e.g., tcpcrypt) to mitigate these weaknesses.

4.5.1 The Case for Authentication and Encryption by Default. This research supports the findings of Caldera et al. and Argyroudis et al. that IPsec can be used to protect network traffic of embedded devices with an acceptable cost to CPU utilization [CDN00, AVT04]. Note that Section 3.8.1 specifies that SHA-256 and AES-256 were used as the cryptographic algorithms. Both algorithms are highly regarded as secure by the cryptographic community. The selected key sizes make brute force attacks against them infeasible today and represent the highest computational cost an embedded device is likely to encounter. Analysis of the measured CPU utilization (Table 9) finds that when no exploit is used, the CPU cost to run IPsec AH+ESP is only 0.52% utilization higher than without a defense protocol. As embedded devices continue to become more powerful, the relative CPU overhead of authentication and encryption algorithms will naturally continue to diminish.

This research also demonstrates the low resources needed to exploit an unprotected protocol running over TCP/IP. Section 3.8.1 details the hardware specifications for Mallory: a laptop released in 2007 running an open-source tool, Scapy. The TCP/IP attacks implemented against Player for this research are publically available and any application layer protocol that

does not employ some form of strong authentication and encryption is therefore assumed to be equally vulnerable. Because advancements in hardware allow more devices to be capable of authentication and encryption and the difficulty in attacking TCP/IP communication is decreasing, authentication and encryption should become the default rather than the exception for network protocol design.

4.5.2 Scaling: CPU vs. Network Overhead. Performance analysis data collected for a single client-server pair running Player can be used to estimate the effects of scaling on system performance. Consider that the number of independent client-server pairs is increased and that each pair that is added to the system consists of another two devices with the same parameters as the client (Alice) and server (Bob), detailed in Section 3.8. Independent client-server pairs transmit over a shared medium but only communicate between pairs. Assume that each pair independently generates the same workload of Player messages tested in this research. Also, assume that Mallory does not launch exploits. Note that precise analysis of this problem would require network simulation or experimentation, but an estimation of trends is still possible with the data gathered in this research.

Each client-server pair added to the system introduces additional CPU resources that can be used to perform the authentication and encryption algorithms necessary for IPsec to protect communication. Since each client-server pair is independent in its communication, it is estimated that the CPU overhead with respect to IPsec will not increase. In contrast, each client-server pair added to the system does not introduce additional network resources. Since each additional client-server pair consumes additional network resources, the network overhead of IPsec becomes significant.

The maximum independent client-server pairs that the system can support when no exploit is used is estimated as,

$$For\ e = None:\ EstPairs_d = \frac{n_{max}}{n_e} \tag{8}$$

Table 23 lists the estimated maximum number of independent client-server pairs that are supported for each defense protocol. These results are calculated by inputting the maximum network resource (n_{max}, Table 6) and the network load (n_e, Table 10) of the system when no exploit is used into (8). Note the trend that as IPsec adds more protection, the number of independent client-server pairs the system can support decreases. The conclusion is that the scaled system running with no defense protocol supports an estimated 64% more client-server pairs than when it uses IPsec AH+ESP.

Table 23. Estimated Maximum Independent Client-Server Pairs

Factor Level	None	IPsec AH	IPsec ESP	IPsec AH+ESP
None	253	201	183	154

The large performance penalty results from an increase in the average packet size that IPsec introduces. The average packet size (Table 11) increases by 64% when IPsec AH+ESP is employed. A slight difference between the average packet size and estimated maximum pairs is explained by the small number of packets transmitted at the data-link layer, which are not affected by IPsec.

Figure 17 depicts the relationship between the estimated number of client-server pairs and the average packet size. It bears repeating that the number of client-server pairs is only an estimation, and simulated or experimental data is needed to draw more precise conclusions. This trend is substantial because Player communicates using a relatively small average packet size of 94 bytes.
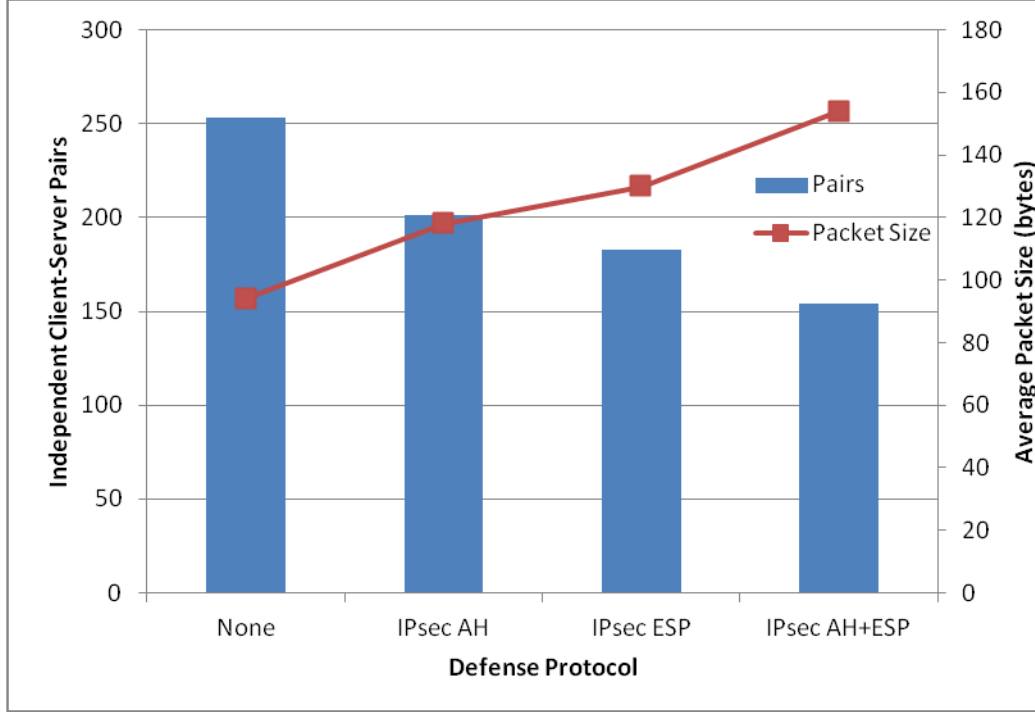
Figure 17. Client-Server Pairs vs. Average Packet Size

As the average packet size for an application increases, the average overhead introduced by IPsec AH+ESP (60 bytes) becomes insignificant. For example, an application with an average packet size of 1500 bytes (the maximum transmission unit of Ethernet) would incur an estimated average packet size increase of only 4% running IPsec AH+ESP. As a result, it is expected that the system would only lose an estimated 4% of the maximum supported pairs.

In summary, results predict that when scaled independently, network overhead, not CPU overhead, is the limiting factor. In addition, the smaller the average packet size of an application is, the higher the relative network overhead of IPsec.

4.5.3 Internet Client-Server Applications. The Player protocol studied in this research is only one of many application layer protocols that run over TCP/IP. However, the vulnerabilities that are demonstrated can be applied to a wider set of protocols. Any TCP/IP protocol that does not employ some form of strong authentication and encryption is potentially vulnerable to the exploits implemented in this research. This research reveals that embedded devices are at least as vulnerable as desktop systems with respect to the network stack when

they communicate using the same insecure protocols. In some ways, embedded devices are more vulnerable as demonstrated by the TCP connection flooding DoS attack. Embedded devices, such as Player robots, acting as servers typically possess fewer computing and memory resources and thus can be overwhelmed more easily.

SSL is the de facto protocol to provide process-to-process security over the Internet for client-server applications. Two of the exploits demonstrated in this research, TCP connection hijacking and TCP reset, compromise weaknesses in the TCP protocol. These weaknesses arise because the TCP header is not authenticated, allowing Mallory to modify information related to the TCP connection without Alice or Bob's knowledge. Because SSL is an application layer protocol, it is not able to authenticate the TCP header, making it vulnerable to DoS attacks that target the TCP connection. Watson demonstrates that TCP reset can be used to attack a TCP/IP protocol that does not authenticate the TCP header [Wat04]. Another disadvantage of SSL is that applications must be modified to support SSL, adding complexity and development cost to these Internet applications.

IPsec is able to authenticate the TCP header, defeating the TCP DoS attacks described in the preceding paragraph, but it provides only machine-to-machine, rather than process-to-process, security. The disadvantage is that IPsec cannot authenticate users, only machines. For example, IPsec cannot distinguish between multiple Player clients on the same machine. In addition, IPsec does not interoperate over Network Address Translation (NAT) which is a popular technique used to extend IPv4 addresses. NAT could be needed in a mobile environment when Player is actually deployed with IPv4.

Tcpcrypt, described in Section 2.3.3, mitigates many of the disadvantages that SSL and IPsec possess. It provides authentication to the TCP header, protection to any application layer protocol, and process-to-process security. By authenticating the sequence number, shown in

Figure 18, tcpcrypt defeats TCP connection hijacking, and by authenticating the flags, tcpcrypt defeats TCP reset. Because tcpcrypt operates in the transport layer, it requires less modification to applications than SSL. Tcpcrypt also interoperates with NAT, allowing Player to utilize this technology if needed. Therefore because of these advantages, tcpcrypt deserves attention from the security community as a mechanism to provide more protection to TCP/IP applications than is currently available.

src port		dst port		
seq no.				(64-bit seq)
ack no.				(64-bit ack)
d.off. flags	window	checksum	urg. ptr.	
options (e.g., SACK)		MAC option		
data (encrypted)				IP length
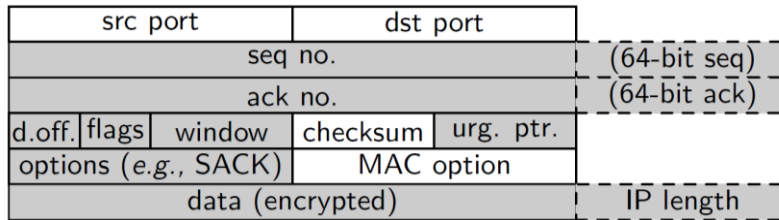
Figure 18. Data Packet Using Tcpcrypt [BHH10].

4.6 Analysis and Results Summary

This chapter analyzes the results of the experiments defined in Chapter 3 to accomplish the research goals of this thesis. A cost function is defined to quantify the performance and security cost associated with running a defense protocol for protection against exploitation. Measured data is input to the cost function to determine the defense protocol with least cost. Under the Likely Exploit Scenario, IPsec AH+ESP is found to be the preferred defense protocol because of its relatively low CPU and network performance costs and its ability to defeat all the exploits implemented in this research. Two additional scenarios are explored to demonstrate the flexibility of the cost function for different use-cases. Results support the case for authentication and encryption of TCP/IP communication by default and highlight potential challenges of using IPsec for systems that must scale to many hosts sending small packets. This research identifies tcpcrypt as a security protocol of interest for client-server applications that communicate over TCP/IP because of its unique ability to avoid some of the disadvantages associated with IPsec and SSL.

V. Conclusions and Recommendations

5.1 Thesis Summary

The following three research goals are presented in Chapter 3 and accomplished in the analysis provided in Chapter 4.

    1) Demonstrate the vulnerability of the Player protocol to network attacks.

This thesis demonstrates that the Player protocol is vulnerable to attacks on all three principles of the CIA security model. Results show that an attacker can eavesdrop on position data sent from the robot (Player server) to the command station (Player client). In addition, Player is vulnerable to man-in-the-middle attacks that allow an attacker to violate the integrity of the position data sent from the robot to the command station. Finally, this research demonstrates that Player is vulnerable to denial-of-service attacks that compromise the availability of the command station to command and control the robot.

    2) Demonstrate the effectiveness of IPsec to secure the Player protocol.

Experimental results demonstrate that IPsec AH is capable of securing Player against the attacks against integrity, and availability implemented in this thesis. IPsec ESP with no optional authentication defeated attacks against confidentiality, integrity, and availability but is not recommended due to published vulnerabilities of its own [Bel96, PaY06]. Results show that IPsec AH+ESP provides the highest level of security because it defeats all of the exploits implemented in this research and has no published weaknesses.

    3) Quantify the cost of IPsec to secure the Player protocol.

Metrics gathered from this research show that mobile devices are well equipped to secure Player-like command and control communication with IPsec. IPsec AH+ESP increased the CPU utilization by just 0.52% and the network load by 22.9Kbps (64.3% increase). Results

70

from the cost function, defined in Section 4.2, show that for the Likely Exploit Scenario, IPsec AH+ESP is the preferred defense protocol because of its ability to defeat exploits and relatively low overhead. For the Confidentiality Free Scenario, IPsec AH is found to be the optimal defense protocol because it defeats integrity and availability exploits implemented in this research. Finally, in the Unlikely Exploit Scenario, using no defense protocol results in the lowest cost because the probability of attack in this scenario is miniscule. The low probability of attack causes the overhead of IPsec in the cost function to become significant.

When results are applied to the wider field of network security, three areas are highlighted. First, this research demonstrates the low capabilities needed for an attacker to compromise an unprotected protocol running over TCP/IP and that modern mobile devices are easily capable of authenticating and encrypting network communication. As a result, this research supports that authentication and encryption become the default rather than the exception for network communication of mobile devices. Second, this research finds that the network overhead, not CPU overhead, of IPsec is expected to be a limiting factor when the system is scaled. Because Player communicates with a low average packet size the network overhead of IPsec is relatively large (63.8% increase compared to no defense protocol). Finally, the TCP denial-of-service attacks implemented in this research are expected to be effective against SSL because it cannot authenticate the TCP header. Tcpcrypt is identified as a protocol of interest that defeats such attacks while avoiding some of the disadvantages associated with SSL and IPsec.

5.2 Recommendations for Future Research

This section describes future research to extend this thesis.

- Performance analysis using a wireless network would be beneficial as this is likely how a Player system would be actually deployed. A wireless environment would

71

make it possible to realistically measure the effects of scaling on system performance when IPsec is deployed. In addition, the attacker's resources could also be scaled to determine how this affects the effectiveness of attacks.

- Power consumption data is excluded from analysis because the data correlated redundantly with the CPU utilization metric in this experimental configuration. The software and hardware design provided in this thesis to characterize the power consumption of a Player device is applicable to wireless environments. Characterizing the power consumption of a CPU and wireless NIC together under different protocols would be of interest. Power consumption data could then be integrated into the cost function developed in Chapter 4.

- The methodology detailed in Chapter 3 of measuring the performance of the system with respect to defense protocols and exploits could be used to detect and identify attacks. A methodology could be developed to create signatures that bind an exploit to a certain set of metric values.

- Performance analysis of SSL and tcpcrypt's ability to secure Player communication would be of interest. When employing SSL, Player could be shown to remain vulnerable to certain denial-of-service attacks. By mitigating these vulnerabilities, the advantage of using tcpcrypt would be demonstrated.

5.3 Final Thoughts

Results of this thesis reiterate and underscore the dangers of performing command and control or any sensitive communication over TCP/IP without authentication and encryption. A fundamental shift away from this type of protocol design must be enacted. Rather than plaintext by default protocols, software engineers must incorporate end-to-end security solutions such as IPsec, SSL, or tcpcrypt, which is currently under review. Modern hardware advances

make such a fundamental change feasible. This shift would significantly reduce the ability for malicious users to perform network attacks.

Appendix A: Player Protocol Wireshark Dissector Source Code

```
#ifdef HAVE_CONFIG_H
# include "config.h"
#endif

#include <epan/packet.h>

/*********** Player Defines *************************/
/***** Message Type Defines ******/
#define PLAYER_MSGTYPE_DATA 1
#define PLAYER_MSGTYPE_CMD 2
#define PLAYER_MSGTYPE_REQ 3
#define PLAYER_MSGTYPE_RESP_ACK 4
#define PLAYER_MSGTYPE_SYNCH 5
#define PLAYER_MSGTYPE_RESP_NACK 6

/***** Device Interface Defines ******/
#define PLAYER_PLAYER_CODE 1
#define PLAYER_POWER_CODE 2
#define PLAYER_GRIPPER_CODE 3
#define PLAYER_POSITION2D_CODE 4

/*** Player Device Interface Subtypes ***/
/* Defined in libplayerinterface\player_interfaces.h */
/** Player:Request Subtypes **/
#define PLAYER_PLAYER_REQ_DEVLIST 1
#define PLAYER_PLAYER_REQ_DRIVERINFO 2
#define PLAYER_PLAYER_REQ_DEV 3
#define PLAYER_PLAYER_REQ_DATA 4
#define PLAYER_PLAYER_REQ_DATAMODE 5
#define PLAYER_PLAYER_REQ_AUTH 7
#define PLAYER_PLAYER_REQ_NAMESERVICE 8
#define PLAYER_PLAYER_REQ_ADD_REPLACE_RULE 10

/** Player:Synch Subtypes **/
#define PLAYER_PLAYER_SYNCH_OK 1
#define PLAYER_PLAYER_SYNCH_OVERFLOW 2

/** Payload Player:Request:Datamode **/
#define PLAYER_DATAMODE_PUSH  1
#define PLAYER_DATAMODE_PULL  2


/*** Position2d Device Interface Subtypes ***/
```

74

```
/*** Position2d:Request Subtypes ***/
#define PLAYER_POSITION2D_REQ_GET_GEOM 1
#define PLAYER_POSITION2D_REQ_MOTOR_POWER 2
#define PLAYER_POSITION2D_REQ_VELOCITY_MODE 3
#define PLAYER_POSITION2D_REQ_POSITION_MODE 4
#define PLAYER_POSITION2D_REQ_SET_ODOM 5
#define PLAYER_POSITION2D_REQ_RESET_ODOM 6
#define PLAYER_POSITION2D_REQ_SPEED_PID 7
#define PLAYER_POSITION2D_REQ_POSITION_PID 8
#define PLAYER_POSITION2D_REQ_SPEED_PROF 9

/** Position2d:Data Subtypes ***/
#define PLAYER_POSITION2D_DATA_STATE 1
#define PLAYER_POSITION2D_DATA_GEOM 2

/** Position2d:Command Subtypes */
#define PLAYER_POSITION2D_CMD_VEL 1
#define PLAYER_POSITION2D_CMD_POS 2
#define PLAYER_POSITION2D_CMD_CAR 3
#define PLAYER_POSITION2D_CMD_VEL_HEAD 4

/*********** Player Defines *************************/


#define PLAYER_PORT 6665

static int proto_player = -1;

/* hf_* variables are used to hold the Wireshark IDs of
 * our header fields; they are filled out when we call
 * proto_register_field_array() in proto_register_player()
 */
static int hf_player_header = -1;
static int hf_player_header_host = -1;
static int hf_player_header_robot = -1;
static int hf_player_header_interface = -1;
static int hf_player_header_index = -1;
static int hf_player_header_type = -1;
static int hf_player_header_subtype = -1;
static int hf_player_header_subtype_player_req = -1;
static int hf_player_header_subtype_player_synch = -1;
static int hf_player_header_subtype_position2d_req = -1;
static int hf_player_header_subtype_position2d_data = -1;
static int hf_player_header_subtype_position2d_cmd = -1;
static int hf_player_header_timestamp = -1;
```

```
static int hf_player_header_sequencenumber = -1;
static int hf_player_header_payloadsize = -1;

static int hf_player_payload = -1;

/* Payload Player:Request:Datamode */
static int hf_player_payload_player_datamode = -1;

/* Payload Player:ResponseAck:Device */
static int hf_player_payload_player_respack_device_name = -1;

/* payload_position2d_data  (also includes payload_position2d_cmd_vel */
static int hf_player_payload_position2d_data_px = -1;
static int hf_player_payload_position2d_data_py = -1;
static int hf_player_payload_position2d_data_pa = -1;

/* payload_position2d_cmd_vel */
static int hf_player_payload_position2d_cmd_vel_vx = -1;
static int hf_player_payload_position2d_cmd_vel_vy = -1;
static int hf_player_payload_position2d_cmd_vel_va = -1;
static int hf_player_payload_position2d_cmd_vel_motorstate = -1;

/* IDs of the subtrees that may be created */
static int ett_player = -1;
static int ett_player_header = -1;
static int ett_player_payload = -1;

/* Enumeration Labels */
static const value_string header_interface_names[] =
{
        {PLAYER_PLAYER_CODE, "Player"},
        {PLAYER_POWER_CODE, "Power"},
        {PLAYER_GRIPPER_CODE, "Gripper"},
        {PLAYER_POSITION2D_CODE, "Position2D"},
        {0, NULL}
};

static const value_string header_messagetype_names[]=
{
        {PLAYER_MSGTYPE_DATA, "Data"},
        {PLAYER_MSGTYPE_CMD, "Command"},
        {PLAYER_MSGTYPE_REQ, "Request"},
        {PLAYER_MSGTYPE_RESP_ACK, "Response-Ack"},
        {PLAYER_MSGTYPE_SYNCH, "Synch"},
        {PLAYER_MSGTYPE_RESP_NACK, "Response-NegAck"},
```

```
        {0, NULL}
};


/******** PLAYER_PLAYER_CODE Device Interface Subtypes **********/
/**** Player:Request, Subtypes ****/
static const value_string header_subtype_player_req_names[]=
{
        {PLAYER_PLAYER_REQ_DEVLIST, "Device List"},
        {PLAYER_PLAYER_REQ_DRIVERINFO, "Driver Info"},
        {PLAYER_PLAYER_REQ_DEV, "Device"},
        {PLAYER_PLAYER_REQ_DATA, "Data"},
        {PLAYER_PLAYER_REQ_DATAMODE, "Datamode"},
        {PLAYER_PLAYER_REQ_AUTH, "Auth"},
        {PLAYER_PLAYER_REQ_NAMESERVICE, "Nameservice"},
        {PLAYER_PLAYER_REQ_ADD_REPLACE_RULE, "Add/Replace Rule"},
        {0, NULL}
};
/**** Player:Synch, Subtypes ****/
static const value_string header_subtype_player_synch_names[]=
{
        {PLAYER_PLAYER_SYNCH_OK, "OK"},
        {PLAYER_PLAYER_SYNCH_OVERFLOW, "Overflow"},
        {0, NULL}
};


/******** PLAYER_PLAYER_CODE Device Interface Subtypes **********/
/**** Position2D:Request, Subtypes ****/
static const value_string header_subtype_position2d_req_names[]=
{
        {PLAYER_POSITION2D_REQ_GET_GEOM, "Get Geometry"},
        {PLAYER_POSITION2D_REQ_MOTOR_POWER, "Motor Power"},
        {PLAYER_POSITION2D_REQ_VELOCITY_MODE, "Velocity Mode"},
        {PLAYER_POSITION2D_REQ_POSITION_MODE, "Position Mode"},
        {PLAYER_POSITION2D_REQ_SET_ODOM, "Set Odom"},
        {PLAYER_POSITION2D_REQ_RESET_ODOM, "Reset Odom"},
        {PLAYER_POSITION2D_REQ_SPEED_PID, "Speed PID"},
        {PLAYER_POSITION2D_REQ_POSITION_PID, "Position PID"},
        {PLAYER_POSITION2D_REQ_SPEED_PROF, "Speed Profile"},
        {0, NULL}
};
/**** Position2D:Data, Subtypes ****/
static const value_string header_subtype_position2d_data_names[]=
{
        {PLAYER_POSITION2D_DATA_STATE, "State"},
        {PLAYER_POSITION2D_DATA_GEOM, "Geometry"},
```

```
        {0, NULL}
};
/**** Position2D:Command, Subtypes ****/
static const value_string header_subtype_position2d_cmd_names[]=
{
        {PLAYER_POSITION2D_CMD_VEL, "Velocity"},
        {PLAYER_POSITION2D_CMD_POS, "Position"},
        {PLAYER_POSITION2D_CMD_CAR, "Car-like"},
        {PLAYER_POSITION2D_CMD_VEL_HEAD, "Heading"},
        {0, NULL}
};


/*************** PAYLOAD Enumeration Labels ****************************/
/* Player:Request:Datamode */
static const value_string payload_player_datamode_names[]=
{
        {PLAYER_DATAMODE_PUSH, "Push"},
        {PLAYER_DATAMODE_PULL, "Pull"},
        {0, NULL}
};


/* Dissector function for the Player Protocol */
/* The Dissector is called in two different cases, one to get a
   summary of the packet (tree==NULL), and one to get details of the packet */
/* tvb: buffer to hold packet data
   pinfo: contains general info about the protocol
   tree: detailed dissection */
static void dissect_player(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{
        /* Offset tracks the location of the current item added to the tree */
        gint offset = 0;

        guint device_interface = 0;
        guint message_type = 0;
        guint message_subtype = 0;
        guint payload_size = 0;

        /* Set column text to protocol name */
    col_set_str(pinfo->cinfo, COL_PROTOCOL, "PLAYER");

        /* Clear out stuff in the info column */
    col_clear(pinfo->cinfo,COL_INFO);

        /* Update the info column with header information */
        device_interface = tvb_get_ntohl(tvb, 8);    /* Defined in tvbuff.h */
```

78

```
message_type = tvb_get_ntohl(tvb, 16);
message_subtype = tvb_get_ntohl(tvb, 20);
payload_size = tvb_get_ntohl(tvb, 36);
col_add_fstr(pinfo->cinfo, COL_INFO, "Inter: %s, Type: %s, Payload Len: %d",
        val_to_str(device_interface, header_interface_names, "Unknown (0x%02x)"),
        val_to_str(message_type, header_messagetype_names, "Unknown (0x%02x)"),
        payload_size
        );

/* When tree != NULL, this is main asking for details of the packet */
if(tree != NULL)
{
        proto_item *player_item = NULL;
        proto_item *player_header_item = NULL;
        proto_item *player_payload_item = NULL;

        proto_tree *player_tree = NULL;
        proto_tree *player_header_tree = NULL;
        proto_tree *player_payload_tree = NULL;

        /* Add a new tree node, label with Player protocol, tvb=data, consume */
        /* from beginning (0) to end (-1). */
        player_item = proto_tree_add_item(tree, proto_player, tvb, 0, -1, FALSE);
        /* Add a Player subtree to the new Tree Node */
        player_tree = proto_item_add_subtree(player_item, ett_player);

        /* Add a Header subtree to the Player Tree */
        /* Headers are 40bytes long */
        player_header_item = proto_tree_add_item(player_tree, hf_player_header, tvb, offset, 40, FALSE);
        player_header_tree = proto_item_add_subtree(player_header_item, ett_player_header);

        /* Add Header items */
        proto_tree_add_item(player_header_tree, hf_player_header_host, tvb, offset, 4, FALSE);
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_robot, tvb, offset, 4, FALSE);
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_interface, tvb, offset, 4, FALSE);
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_index, tvb, offset, 4, FALSE);
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_type, tvb, offset, 4, FALSE);
        offset+=4;

        /* The Header Message Subtype is defined based on the Device Interface and Message Type*/
        /* of that packet */
```

```
                        switch(device_interface)
                        {
                                case PLAYER_PLAYER_CODE:
                                        switch(message_type)
                                        {
                                                case PLAYER_MSGTYPE_REQ:
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_player_req, tvb, offset,
4, FALSE);
                                                        break;
                                                case PLAYER_MSGTYPE_SYNCH:
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_player_synch, tvb,
offset, 4, FALSE);
                                                        break;
                                                case PLAYER_MSGTYPE_RESP_ACK: /* Uses same subtypes as Request Message Type */
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_player_req, tvb, offset,
4, FALSE);
                                                        break;
                                                default: /* Use generic subtype */
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype, tvb, offset, 4, FALSE);
                                                        break;
                                        }
                                        break;

                                case PLAYER_POWER_CODE:
                                        break;

                                case PLAYER_GRIPPER_CODE:
                                        break;

                                case PLAYER_POSITION2D_CODE:
                                        switch(message_type)
                                        {
                                                case PLAYER_MSGTYPE_REQ:
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_position2d_req, tvb,
offset, 4, FALSE);
                                                        break;
                                                case PLAYER_MSGTYPE_DATA:
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_position2d_data, tvb,
offset, 4, FALSE);
                                                        break;
                                                case PLAYER_MSGTYPE_CMD:
                                                        proto_tree_add_item(player_header_tree, hf_player_header_subtype_position2d_cmd, tvb,
offset, 4, FALSE);
                                                        break;
                                                default: /* Use generic subtype */
```

```
                                                proto_tree_add_item(player_header_tree, hf_player_header_subtype, tvb, offset, 4, FALSE);
                                                break;
                                }
                                break;
                        default: /* If device interface is other than one defined above, display generic version */
                                proto_tree_add_item(player_header_tree, hf_player_header_subtype, tvb, offset, 4, FALSE);
                                break;
                }
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_timestamp, tvb, offset, 8, FALSE);
        offset+=8; /* Double = 8 bytes */
        proto_tree_add_item(player_header_tree, hf_player_header_sequencenumber, tvb, offset, 4, FALSE);
        offset+=4;
        proto_tree_add_item(player_header_tree, hf_player_header_payloadsize, tvb, offset, 4, FALSE);
        offset+=4;

        /* Add text summary for dissection window */
        proto_item_append_text(player_item, ", Inter: %s, Type: %s, Payload Len: %d",
val_to_str(device_interface, header_interface_names, "Unknown (0x%02x)"),
                val_to_str(message_type, header_messagetype_names, "Unknown (0x%02x)"),
                payload_size
                );

        /* If a Payload is present, add a Payload subtree to the Player Tree */
        /* The Payload comprises whatever data (if any) is present after the 40 byte header */
        if(tvb_length(tvb) > 40) /* Defined in tvbuff.h */
        {
                player_payload_item = proto_tree_add_item(player_tree, hf_player_payload, tvb, 40, -1, FALSE);
                player_payload_tree = proto_item_add_subtree(player_payload_item, ett_player_payload);


                switch(device_interface)
                {
                        case PLAYER_PLAYER_CODE:
                                switch(message_type)
                                {
                                        case PLAYER_MSGTYPE_REQ:
                                                switch(message_subtype)
                                                {
                                                        case PLAYER_PLAYER_REQ_DEVLIST:
                                                                break;

                                                        case PLAYER_PLAYER_REQ_DRIVERINFO:
                                                                break;
```

81

```
                                                  case PLAYER_PLAYER_REQ_DEV: /* Uses similar data format to header */
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_host, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_robot, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_interface, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                          proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                          offset+=4;
                                                  break;

                                          case PLAYER_PLAYER_REQ_DATA:
                                                  break;

                                          case PLAYER_PLAYER_REQ_DATAMODE:
                                                  proto_tree_add_item(player_payload_tree,
hf_player_payload_player_datamode, tvb, offset, 4, FALSE);

                                                  offset+=4;
                                                  break;

                                          case PLAYER_PLAYER_REQ_AUTH:
                                                  break;

                                          case PLAYER_PLAYER_REQ_NAMESERVICE:
                                                  break;

                                          case PLAYER_PLAYER_REQ_ADD_REPLACE_RULE:
                                                  break;
                                  }
                          break;

                  case PLAYER_MSGTYPE_RESP_ACK:
```

```
                                                switch(message_subtype)
                                                {
                                                        case PLAYER_PLAYER_REQ_DEV:  /* Uses similar data format to header
*/
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_host, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_robot, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_interface, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_header_index, tvb, offset, 4, FALSE);

                                                                offset+=4;
                                                                proto_tree_add_item(player_payload_tree,
hf_player_payload_player_respack_device_name, tvb, offset, -1, FALSE);

                                                                /* offset+=4;  Rest of payload is a string containing the
device name */

                                                                break;
                                                }
                                                break;
                                }
                                break;

                        case PLAYER_POWER_CODE:
                                break;

                        case PLAYER_GRIPPER_CODE:
                                break;

                        case PLAYER_POSITION2D_CODE:
                                switch(message_type)
                                {
                                        case PLAYER_MSGTYPE_DATA:
```

83

```
                                        /* Payload: player_position2d_data */
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_data_px,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_data_py,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_data_pa,
tvb, offset, 8, FALSE);
                                        offset+=8;

                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_vx,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_vy,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_va,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree,
hf_player_payload_position2d_cmd_vel_motorstate, tvb, offset, 4, FALSE);
                                        offset+=4;
                                        break;
                                    case PLAYER_MSGTYPE_CMD:
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_vx,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_vy,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree, hf_player_payload_position2d_cmd_vel_va,
tvb, offset, 8, FALSE);
                                        offset+=8;
                                        proto_tree_add_item(player_payload_tree,
hf_player_payload_position2d_cmd_vel_motorstate, tvb, offset, 4, FALSE);
                                        offset+=4;
                                        break;

                            }
                            break;
                    }
                }
            }
}
```

```
/* Registers the protocol with Wireshark */
void proto_register_player(void)
{
        /* A header field is something you can search/filter on.
         *
         * We create a structure to register our fields. It consists of an
         * array of hf_register_info structures, each of which are of the format
         * {&(field id), {name, abbrev, type, display, strings, bitmask, blurb, HFILL}}.
         */
        static hf_register_info hf[] =
        {
                /* HEADER SECTION */
                /* FT_* defined in ftypes.h */
                {
                        &hf_player_header,
                        {
                                "Header", "player.header",
                                FT_NONE, BASE_NONE,
                                NULL, 0x0, "Player Header", HFILL
                        }
                },
                {
                        &hf_player_payload,
                        {
                                "Payload", "player.payload",
                                FT_NONE, BASE_NONE,
                                NULL, 0x0, "Player Payload", HFILL
                        }
                },
                {
                        &hf_player_header_host,
                        {
                                "Device Host", "player.devicehost",
                                FT_IPv4, BASE_NONE,
                                NULL, 0x0, NULL, HFILL
                        }
                },
                {
                        &hf_player_header_robot,
                        {
                                "Device Robot", "player.devicerobot",
                                FT_UINT32, BASE_DEC,
                                NULL, 0x0, NULL, HFILL
```

```
                }
        },
        {
                &hf_player_header_interface,
                {
                        "Device Interface", "player.interface",
                        FT_UINT32, BASE_DEC,
                        header_interface_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_index,
                {
                        "Device Index", "player.index",
                        FT_UINT32, BASE_DEC,
                        NULL, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_type,
                {
                        "Type", "player.type",
                        FT_UINT32, BASE_DEC,
                        header_messagetype_names, 0x0, NULL, HFILL
                }
        },
        { /* Need to create multiple messagesubtype definitions. Each Device Interface */
          /* Defines its own message subtypes for each message type defines */
                &hf_player_header_subtype,  /* Generic message subtype def */
                {
                        "Subtype", "player.subtype",
                        FT_UINT32, BASE_DEC,
                        NULL, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_subtype_player_req,
                {
                        "Subtype", "player.subtype.player.req",
                        FT_UINT32, BASE_DEC,
                        header_subtype_player_req_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_subtype_player_synch,
```

86

```
                {
                        "Subtype", "player.subtype.player.synch",
                        FT_UINT32, BASE_DEC,
                        header_subtype_player_synch_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_subtype_position2d_req,
                {
                        "Subtype", "player.subtype.position2d.req",
                        FT_UINT32, BASE_DEC,
                        header_subtype_position2d_req_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_subtype_position2d_data,
                {
                        "Subtype", "player.subtype.position2d.data",
                        FT_UINT32, BASE_DEC,
                        header_subtype_position2d_data_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_subtype_position2d_cmd,
                {
                        "Subtype", "player.subtype.position2d.cmd",
                        FT_UINT32, BASE_DEC,
                        header_subtype_position2d_cmd_names, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_timestamp,
                {
                        "Timestamp", "player.timestamp",
                        FT_DOUBLE, BASE_NONE,
                        NULL, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_header_sequencenumber,
                {
                        "Sequence Number", "player.sequencenumber",
                        FT_UINT32, BASE_DEC,
                        NULL, 0x0, NULL, HFILL
                }
```

```
        },
        {
                &hf_player_header_payloadsize,
                {
                        "Payload Size", "player.payloadsize",
                        FT_UINT32, BASE_DEC,
                        NULL, 0x0, NULL, HFILL
                }
        },


/********************** PAYLOAD SECTION **********************************************/

/****** Player:Request:Datamode ********/
        {
                &hf_player_payload_player_datamode,
                {
                        "Data Mode", "player.payload.player.datamode",
                        FT_UINT32, BASE_NONE,
                        payload_player_datamode_names, 0x0, NULL, HFILL
                }
        },
/****** Player:Response-Ack:Device ********/
        {
                &hf_player_payload_player_respack_device_name,
                {
                        "Name", "player.payload.player.respack.device.name",
                        FT_STRINGZ, BASE_NONE,
                        NULL, 0x0, NULL, HFILL
                }
        },

/******************** DATA TYPE SECTION *********************/
/* Position2d: player_position2d_data */
        {
                &hf_player_payload_position2d_data_px,
                {
                        "PositionX (m)", "player.payload.position2d_data.px",
                        FT_DOUBLE, BASE_NONE,
                        NULL, 0x0, NULL, HFILL
                }
        },
        {
                &hf_player_payload_position2d_data_py,
                {
```

```
                "PositionY (m)", "player.payload.position2d_data.py",
                FT_DOUBLE, BASE_NONE,
                NULL, 0x0, NULL, HFILL
        }
},
{
        &hf_player_payload_position2d_data_pa,
        {
                "PositionA (rad)", "player.payload.position2d_data.pa",
                FT_DOUBLE, BASE_NONE,
                NULL, 0x0, NULL, HFILL
        }
},


/********************* CMD TYPE SECTION *********************/

/* Position2d: player_position2d_cmd_vel */
{
        &hf_player_payload_position2d_cmd_vel_vx,
        {
                "VelocityX (m/s)", "player.payload.position2d_cmd_vel.vx",
                FT_DOUBLE, BASE_NONE,
                NULL, 0x0, NULL, HFILL
        }
},
{
        &hf_player_payload_position2d_cmd_vel_vy,
        {
                "VelocityY (m/s)", "player.payload.position2d_cmd_vel.vy",
                FT_DOUBLE, BASE_NONE,
                NULL, 0x0, NULL, HFILL
        }
},
{
        &hf_player_payload_position2d_cmd_vel_va,
        {
                "VelocityA (rad/s)", "player.payload.position2d_cmd_vel.va",
                FT_DOUBLE, BASE_NONE,
                NULL, 0x0, NULL, HFILL
        }
},
{
        &hf_player_payload_position2d_cmd_vel_motorstate,
        {
```

```
                                "Motor State", "player.payload.position2d_cmd_vel.motorstate",
                                FT_UINT32, BASE_DEC,
                                NULL, 0x0, NULL, HFILL
                        }
                }



        };

        /* Setup protocol subtree array */
        static gint *ett[] =
        {
                &ett_player,
                &ett_player_header,
                &ett_player_payload
        };

        /* Registers protocol. Format: (Name, Short Name, Abbrev) */
    proto_player = proto_register_protocol ("Player Protocol", "Player", "player");

        proto_register_field_array(proto_player, hf, array_length(hf));
        proto_register_subtree_array(ett, array_length(ett));
}

/* Creates a dissector handle for main program to call */
void proto_reg_handoff_player(void)
{
    static dissector_handle_t player_handle;

        /* Create a dissector handle associate with the player protocol and with
        a routine to be called to dissect it*/
    player_handle = create_dissector_handle(dissect_player, proto_player);

        /*Associate the player_handle with a TCP port number so that the main
        program will know to call us when TCP traffic arrives on that port*/
    dissector_add_uint("tcp.port", PLAYER_PORT, player_handle);
}
```

Appendix B: IPsec Security Associations

```
########################################################################
## IPsec Configuration for Player Defense System
## John Hagen
## Masters in Cyber Operations AFIT/ENG AFRL/RYWC
########################################################################


############ Flush the SAD and SPD ###############################
flush;
spdflush;


########## Security Associations AH SHA256-256 bit key #############
add 192.168.1.2 192.168.1.3 ah 0x200 -A hmac-sha256
     0x8d375a74b4a2c70d36dc9c6de2179c4493f30034ef3c3682afb6be2b60bf42e9;
add 192.168.1.3 192.168.1.2 ah 0x300 -A hmac-sha256
     0xb037b7c2a619fb0987bff4708eef2fb328c79aef1c26fddd46f2138a493c8708;


########## Security Associations ESP AES-256 bit key ##############
add 192.168.1.2 192.168.1.3 esp 0x201 -E aes-cbc
     0x25ea0b76e21f20acab36da6642feb056fe98f14439b02db25091b13a5b85a75b;
add 192.168.1.3 192.168.1.2 esp 0x301 -E aes-cbc
     0x5e3bebefdaea58e98433e7b7824e6950756012e81aadd38509c5fb5cc7c3bda5;
## To add authentication directly to ESP rather than use in conjunction with
AH add:
## -A hmac-sha256
## <SHA256 key>


########## Security Policies - AH Only ###########################
## Require IPsec for all IP communication
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P out ipsec
##     ah/transport//require;
##
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P in ipsec
##     ah/transport//require;


########## Security Policies - ESP Only ##########################
## Require IPsec for all IP communication
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P out ipsec
##     esp/transport//require;
##
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P in ipsec
##     esp/transport//require;



########## Security Policies - AH+ESP ############################
## Require IPsec for all IP communication
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P out ipsec
##     esp/transport//require
##     ah/transport//require;
##
##spdadd 255.255.255.255/0 255.255.255.255/0 any -P in ipsec
##     esp/transport//require
##     ah/transport//require;
```

Appendix C: Configuration Instructions

Alice:

1) Download Ubuntu Desktop 11.10 32-bit ISO
    a. http://www.ubuntu.com/download/ubuntu/download
2) Burn the ISO file to a CD and boot with the CD to install Ubuntu 11.10.
3) Install Ubuntu 11.10
    a. Select Download updates while installing.
    b. Do not install third party applications.
4) Open the Update Manager
    a. Install all important security updates.
    b. Do not update to a newer version of Ubuntu if available.
    c. Restart the computer.
5) Open the Ubuntu Software center
    a. Search for and install Synaptic Package Manager
6) Open Synaptic Package Manager and install the following packages:
    a. robot-player   3.0.2
    b. ipsec-tools   0.7.3
    c. ckermit   211-14   (To interact with the Overo Console via USB)
7) Open Terminal
    a. sudo nautilus
    b. Open /etc/network/interfaces with gedit. Add the following lines:
        auto eth0
        iface eth0 inet static
        address 192.168.1.2
        netmask 255.255.255.0

Mallory:

1. Download Ubuntu Desktop 11.10 32-bit iso
   a. http://www.ubuntu.com/download/ubuntu/download
2. Burn the ISO file to a CD and boot with the CD to install Ubuntu 11.10.
3. Install Ubuntu 11.10
   a. Select Download updates while installing.
   b. Do not install third party applications.
4. Open the Update Manager
   a. Install all important security updates.
   b. Do not update to a newer version of Ubuntu if available.
   c. Restart the computer.
5. Open the Ubuntu Software center
   a. Search for and install Synaptic Package Manager
6. Open the Synaptic Package Manager and install the following packages:
   a. python-scapy 2.1.0
   b. python-psyco 1.6
   c. spe   0.8.4   (Stani's Python Editor)
7. Open Terminal
   a. sudo nautilus
   b. Open /etc/network/interfaces with gedit. Add the following lines:
      auto eth0
      iface eth0 inet static
      address 192.168.1.4
      netmask 255.255.255.0

Bob:

I. BUILDING OVERO OPEN EMBEDDED IMAGE

Guide: http://gumstix.org/software-development/open-embedded/61-using-the-open-embedded-build-system.html

1) Build a new virtual machine with the Ubuntu 10.10 x86 ISO file to act as the development laptop.
    a. http://releases.ubuntu.com/10.10/ubuntu-10.10-desktop-i386.iso
2) Once booted, use the Update Manager to update the default packages. Do not upgrade to Ubuntu 11.04 or other versions.
3) Open the synaptic package manager and select the following packages for install:
    a. git
    b. subversion
    c. gcc
    d. build-essential
    e. help2man
    f. diffstat
    g. texi2html
    h. texinfo
    i. libncurses5-dev
    j. cvs
    k. gawk
    l. python2.7-dev
    m. python-pysqlite2
    n. unzip
    o. chrpath
    p. ccache
    q. python-psyco
4) sudo dpkg-reconfigure dash
    a. Answer ***No*** when asked whether you want to install dash as /bin/sh.

5) mkdir -p ~/overo-oe
6) cd ~/overo-oe

7) git clone git://gitorious.org/gumstix-oe/mainline.git org.openembedded.dev
8) cd org.openembedded.dev
9) git checkout --track -b overo-2011.03 origin/overo-2011.03

10) cd ~/overo-oe
11) git clone git://git.openembedded.org/bitbake bitbake
12) cd bitbake
13) git checkout 1.12.0

14) cd ~/overo-oe

15) cp -r org.openembedded.dev/contrib/gumstix/build .

16) cp ~/.bashrc ~/bashrc.bak

17) cat ~/overo-oe/build/profile >> ~/.bashrc

18) Close the Terminal window and open a new one.

19) bitbake omap3-console-image

20) The Overo file system is built at: ~/overo-oe/tmp/deploy/glibc/images/overo/omap3-console-image-overo.tar.bz2

21) The Overo OE Linux Kernel is built at: ~/overo-oe/tmp/deploy/glibc/images/overo/uImage-overo.bin

II. RECONFIGURING THE OVERO KERNEL TO INCLUDE IPSEC SUPPORT

Guide: http://www.ipsec-howto.org/x299.html

1) On the development laptop:

2) cd ~/overo-oe

3) mkdir -p ./user.collection/recipes

4) cp -r ./org.openembedded.dev/recipes/linux /home/jhagen/overo-oe/user.collection/recipes

    a. (bitbake looks at user.collection first. org.embedded.dev hold the original copy)

5) cd ~/overo-oe/tmp/work/overo-angstrom-linux-gnueabi/linux-omap3<kernel version>/git

6) make menuconfig ARCH=arm

    a. Networking

        i. Network options

            1. [*] IP: AH Transformation

            2. [*] IP: ESP Transformation

            3. [*] IP: IPsec transport mode

    b. Cryptographic API

        i. [*] Null algorithms

        ii. [*] HMAC support

        iii. [*] MD5 digest algorithm

        iv. [*] SHA1 digest algorithm

        v. [*] AES cipher algorithm

        vi. [*] DES and Triple DES EDE algorithms

    c. Exit

    d. Save: Yes

7) ls –al

    a. Check that date was made today

8) less .config

    a. Scroll to #Networking options and confirm:

i. CONFIG_INET_AH=y

ii. CONFIG_INET_ESP=y

9) cp ./.config ~/overo-oe/user.collection/recipes/linux/linux-omap3/overo/defconfig

10) cd ~/overo

11) bitbake –c clean linux-omap3

12) bitbake –c build linux-omap3

13) The Overo OE Linux Kernel is built at: ~/overo-oe/tmp/deploy/glibc/images/overo/uImage-overo.bin

III. PARTITIONING BOOTABLE SD CARD FOR OVERO IMAGE

Guide: http://gumstix.org/create-a-bootable-microsd-card.html

Guide: http://gumstix.org/how-to/70-writing-images-to-flash.html

IV. DEPLOYING OVERO IMAGE

1) On the development laptop:

2) Delete the current file structure, if any, on the EXT3 partition of the micro SD card
   a. sudo nautilus
   b. Edit > Preferences > Behavior > Check Include a Delete command that bypasses Trash
   c. Select rootfs
   d. Select all files > Right Click > Delete

3) Copy the contents of ~/overo-oe/tmp/deploy/glibc/images/overo/omap3-console-image-overo.tar.bz2 into the rootfs partition of the micro SD card.

4) On the micro SD card FAT partition:
   a. Delete uImage
   b. Copy uImage-<kernel version>-overo.bin into /
   c. Rename uImage-<kernel version>-overo.bin to uImage

V. BOOTING OVERO IMAGE CONSOLE

1) Power off the Overo board.

2) Insert the newly created micro SD card into the micro SD slot of the Overo board.

3) Connect a USB cable between the "Console" mini USB B port on the Overo board and the development laptop with ckermit installed.

4) On the development laptop create a file called overo_serial.cfg

   set line /dev/ttyUSB0(Note: 0 might changed)

   set flow-control none

   set carrier-watch off

   set speed 115200

   set reliable

    fast

    set prefixing all

    set file type bin

    set rec pack 4096

    set send pack 4096

    set window 5

    connect

5) Open a terminal and type:
  a. kermit
     i. take overo_serial.cfg

6) Power on the Overo board. You should see the boot sequence displayed on the terminal. It will finish with a prompt to login.

7) Enter "root" as the username to log in.

8) To exit kermit:
  a. ctrl-/-c
  b. Type: exit

VI. COMPILING EXTRA PACKAGES FOR OVERO IMAGE

1) After the first console image has been built, the environment is set up to build extra packages.

2) On the development laptop:
  a. bitbake <package name>

3) Packages will be built in: /overo-oe/tmp/deploy/glibc/ipk/armv7a

4) Copy the packages onto the Overo EXT3 partition
  a. sudo scp ./<package name>.ipk <overo IP address>:/home/root

5) On the Overo console, install the package
  a. opkg install ./<package name>

VII. SET STATIC IP ADDRESS

1) Edit /etc/network/interfaces. Add the following lines:
    auto eth0

    iface eth0 inet static

    address 192.168.1.3

    netmask 255.255.255.0

    gateway 192.168.1.1

2) Note: A default gateway is added because one attack used against Bob relies on him thinking he can route traffic to IP addresses outside his subnet.

VIII. COMPILING PLAYER FOR OVERO IMAGE

Guide:
http://playerstage.sourceforge.net/wiki/Cross_Compile_Player_with_Openembedded_and_BitBake

1) On the development laptop:
2) Create the following folder: /overo-oe/org.openembedded.dev/recipes/player
3) Save the following as /overo-oe/org.openembedded.dev/recipes/player/player.bb (Modified from the Guide)

```
DESCRIPTION = "Cross-platform robot device interface and server"
LICENSE = "GPLv2+ and LGPLv2+"
HOMEPAGE = "http://playerstage.sourceforge.net"
DEPENDS = "libtool"
PN = "player"
PV = 3.0.2
PR = "r0"
SRC_URI = "http://iweb.dl.sourceforge.net/project/playerstage/Player/${PV}/player-${PV}.tar.gz"
SRC_URI[md5sum] = "b92b5ea028e6bfc49351849f420167db"
S="${WORKDIR}/player-${PV}"

inherit pkgconfig

do_configure () {
    cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_EXAMPLES=OFF -DBUILD_DOCUMENTATION=OFF -
DBUILD_EXAMPLES=OFF -DBUILD_PLAYERCC=OFF -DBUILD_PLAYERCC_BOOST=OFF\
-DBUILD_PYTHONC_BINDINGS=OFF -DBUILD_SHARED_LIBS=OFF -DBUILD_UTILS=OFF -
DBUILD_UTILS_LOGSPLITTER=OFF -DBUILD_UTILS_PLAYERCAM=OFF \
-DBUILD_UTILS_PLAYERJOY=OFF -DBUILD_UTILS_PLAYERNAV=OFF -DBUILD_UTILS_PLAYERPRINT=OFF
-DBUILD_UTILS_PLAYERPROP=OFF -DBUILD_UTILS_PLAYERV=OFF \
-DBUILD_UTILS_PLAYERVCR=OFF -DBUILD_UTILS_PLAYERWRITEMAP=OFF -DBUILD_UTILS_PMAP=OFF -
DBUILD_UTILS_XMMS=OFF -DENABLE_DRIVER_ACCEL_CALIB=OFF \
-DENABLE_DRIVER_ACR120U=OFF -DENABLE_DRIVER_ACTS=OFF -DENABLE_DRIVER_AIOTOSONAR=OFF
-DENABLE_DRIVER_ALSA=OFF -DENABLE_DRIVER_AMCL=OFF \
-DENABLE_DRIVER_AMTECM5=OFF -DENABLE_DRIVER_AMTECPOWERCUBE=OFF -
DENABLE_DRIVER_AODV=OFF -DENABLE_DRIVER_ARTOOLKITPLUS=OFF -
DENABLE_DRIVER_BLOBTODIO=OFF \
-DENABLE_DRIVER_BLOBTRACKER=OFF -DENABLE_DRIVER_BUMPER2LASER=OFF -
DENABLE_DRIVER_BUMPERSAFE=OFF -DENABLE_DRIVER_BUMPERTODIO=OFF -
DENABLE_DRIVER_CAMERA1394=OFF \
-DENABLE_DRIVER_CAMERACOMPRESS=OFF -DENABLE_DRIVER_CAMERAUNCOMPRESS=OFF -
DENABLE_DRIVER_CAMERAUVC=OFF -DENABLE_DRIVER_CAMERAV4L=OFF -
DENABLE_DRIVER_CAMERAV4L2=OFF \
-DENABLE_DRIVER_CAMFILTER=OFF -DENABLE_DRIVER_CANONVCC4=OFF -
DENABLE_DRIVER_CLODBUSTER=OFF -DENABLE_DRIVER_CMDSPLITTER=OFF -
DENABLE_DRIVER_CMUCAM2=OFF \
-DENABLE_DRIVER_CMVISION=OFF -DENABLE_DRIVER_CREATE=ON -DENABLE_DRIVER_CVCAM=OFF -
DENABLE_DRIVER_DEADSTOP=OFF -DENABLE_DRIVER_DIOCMD=OFF -
DENABLE_DRIVER_DIODELAY=OFF \
-DENABLE_DRIVER_DIOLATCH=OFF -DENABLE_DRIVER_DUMMY=OFF -
DENABLE_DRIVER_EEDHCONTROLLER=OFF -DENABLE_DRIVER_EPUCK=OFF -
DENABLE_DRIVER_ER1=OFF -DENABLE_DRIVER_ERRATIC=OFF \
-DENABLE_DRIVER_FAKELOCALIZE=OFF -DENABLE_DRIVER_FESTIVAL=OFF -
DENABLE_DRIVER_FLEXIPORT=OFF -DENABLE_DRIVER_FLOCKOFBIRDS=OFF -
DENABLE_DRIVER_GARCIA=OFF \
```

```
-DENABLE_DRIVER_GARMINNMEA=OFF -DENABLE_DRIVER_GBXGARMINACFR=OFF -
DENABLE_DRIVER_GBXSICKACFR=OFF -DENABLE_DRIVER_SCENARIOIZE=OFF -
DENABLE_DRIVER_GOTO=OFF \
-DENABLE_DRIVER_GRIDMAP=OFF -DENABLE_DRIVER_GRIPCMD=OFF -
DENABLE_DRIVER_HOKUYO_AIST=OFF -DENABLE_DRIVER_IMAGESEQ=OFF -
DENABLE_DRIVER_INHIBITOR=OFF \
-DENABLE_DRIVER_INSIDEM300=OFF -DENABLE_DRIVER_ISENSE=OFF -DENABLE_DRIVER_IWSPY=OFF -
DENABLE_DRIVER_KARTOWRITER=OFF -DENABLE_DRIVER_KHEPERA=OFF \
-DENABLE_DRIVER_LASERBAR=OFF -DENABLE_DRIVER_LASERBARCODE=OFF -
DENABLE_DRIVER_LASERCSPACE=OFF -DENABLE_DRIVER_LASERCUTTER=OFF -
DENABLE_DRIVER_LASERPOSEINTERPOLATOR=OFF \
-DENABLE_DRIVER_LASERPTZCLOUD=OFF -DENABLE_DRIVER_LASERRESCAN=OFF -
DENABLE_DRIVER_LASERSAFE=OFF -DENABLE_DRIVER_LASERTORANGER=OFF -
DENABLE_DRIVER_LASERVISUALBARCODE=OFF \
-DENABLE_DRIVER_LASERVISUALBW=OFF -DENABLE_DRIVER_LINUXJOYSTICK=OFF -
DENABLE_DRIVER_LINUXWIFI=OFF -DENABLE_DRIVER_LOCALBB=OFF -
DENABLE_DRIVER_MAPCSPACE=OFF \
-DENABLE_DRIVER_MAPFILE=OFF -DENABLE_DRIVER_MAPSCALE=OFF -DENABLE_DRIVER_MBICP=OFF -
DENABLE_DRIVER_MICA2=OFF -DENABLE_DRIVER_MICROSTRAIN=OFF -
DENABLE_DRIVER_MOTIONMIND=OFF \
-DENABLE_DRIVER_MRICP=OFF -DENABLE_DRIVER_ND=OFF -DENABLE_DRIVER_NIMU=OFF -
DENABLE_DRIVER_NOMAD=OFF -DENABLE_DRIVER_OBOT=OFF -
DENABLE_DRIVER_OCEANSERVER=OFF \
-DENABLE_DRIVER_P2OS=OFF -DENABLE_DRIVER_PASSTHROUGH=OFF -
DENABLE_DRIVER_PBSLASER=OFF -DENABLE_DRIVER_PHIDGETACC=OFF -
DENABLE_DRIVER_PHIDGETIFK=OFF \
-DENABLE_DRIVER_PHIDGETRFID=OFF -DENABLE_DRIVER_PORTIO=OFF -
DENABLE_DRIVER_POSTGIS=OFF -DENABLE_DRIVER_PTU46=OFF -
DENABLE_DRIVER_RANGERPOSEINTERPOLATOR=OFF \
-DENABLE_DRIVER_RANGERTODIO=OFF -DENABLE_DRIVER_RANGERTOLASER=OFF -
DENABLE_DRIVER_RCORE_XBRIDGE=OFF -DENABLE_DRIVER_READLOG=OFF -
DENABLE_DRIVER_REB=OFF \
-DENABLE_DRIVER_RELAY=OFF -DENABLE_DRIVER_RFLEX=OFF -DENABLE_DRIVER_ROBOTEQ=OFF -
DENABLE_DRIVER_ROBOTINO=OFF -DENABLE_DRIVER_ROBOTRACKER=OFF \
-DENABLE_DRIVER_ROOMBA=OFF -DENABLE_DRIVER_RS4LEUZE=OFF -DENABLE_DRIVER_RT3XXX=OFF
-DENABLE_DRIVER_SEGWAYRMP=OFF -DENABLE_DRIVER_SEGWAYRMP400=OFF \
-DENABLE_DRIVER_SERIALSTREAM=OFF -DENABLE_DRIVER_SERIO=OFF -
DENABLE_DRIVER_SERVICE_ADV_MDNS=OFF -DENABLE_DRIVER_SHAPETRACKER=OFF -
DENABLE_DRIVER_SICKLDMRS=OFF \
-DENABLE_DRIVER_SICKLMS200=OFF -DENABLE_DRIVER_SICKLMS400=OFF -
DENABLE_DRIVER_SICKNAV200=OFF -DENABLE_DRIVER_SICKRFI341=OFF -
DENABLE_DRIVER_SICKS3000=OFF \
-DENABLE_DRIVER_SIMPLESHAPE=OFF -DENABLE_DRIVER_SKYETEKM1=OFF -
DENABLE_DRIVER_SND=OFF -DENABLE_DRIVER_SONARTORANGER=OFF -
DENABLE_DRIVER_SONYEVID30=OFF \
-DENABLE_DRIVER_SPHERE=OFF -DENABLE_DRIVER_SPHEREPTZ=OFF -
DENABLE_DRIVER_SPHINX2=OFF -DENABLE_DRIVER_SR3000=OFF -DENABLE_DRIVER_STALLTODIO=OFF
-DENABLE_DRIVER_STATGRAB=OFF \
-DENABLE_DRIVER_STOC=OFF -DENABLE_DRIVER_SUPPRESSOR=OFF -
DENABLE_DRIVER_SWISSRANGER=OFF -DENABLE_DRIVER_TCPSTREAM=OFF -
DENABLE_DRIVER_UNICAPIMAGE=OFF \
-DENABLE_DRIVER_UPCBARCODE=OFF -DENABLE_DRIVER_VEC2MAP=OFF -
DENABLE_DRIVER_VELCMD=OFF -DENABLE_DRIVER_VFH=OFF -DENABLE_DRIVER_VIDEOCANNY=OFF \
-DENABLE_DRIVER_VMAPFILE=OFF -DENABLE_DRIVER_WAVEFRONT=OFF -
DENABLE_DRIVER_WBR914=OFF -DENABLE_DRIVER_WRITELOG=OFF -DENABLE_DRIVER_XSENSMT=OFF
\
-DENABLE_DRIVER_YARPIMAGE=OFF -DENABLE_DRIVER_SPEECHCMD=OFF -
DENABLE_DRIVER_CAMERAGST=OFF .
}
```

```
do_compile() {

    oe_runmake

}

do_install() {

    oe_runmake install DESTDIR=${D}

}

FILES_${PN} = "/usr/bin/player \
    /usr/bin/playerinterfacegen \
    /usr/bin/playerxdrgen \
    /usr/include/* \
    /usr/lib/* \
    /usr/share/*"
```

4) bitbake player
5) Copy the following packages from ~/overo-oe/tmp/deploy/glibc/ipk/armv7a to the Overo file system:
      a. libstdc++6_4.3.3-r25.2.6_armv7a.ipk
      b. libxml2_2.7.8-r9.1.6_armv7a.ipk
      c. libjpeg8_8b-r1.6_armv7a.ipk
      d. player_3.0.2-r0.6_armv7a.ipk
6) Create a file called overo_create.cfg and copy it the Overo file system:

```
driver
(
  name "create"
  provides ["position2d:0" "power:0" "bumper:0" "ir:0" "opaque:0"]
  port "/dev/ttyO0"
  safe 0
)
```

7) On the Overo console for each of the three packages above type:
      a. opkg install ./<package name>
8) To run player server:
      a. player overo_create.cfg

IX. DATALOGGING CPU USAGE USING SYSSTAT

sar Man page: http://linux.die.net/man/1/sar

1) Install the sysstat 9.0.6 (see COMPILING EXTRA PACKAGES FOR OVERO IMAGE)
2) To print CPU utilization every 1 second. Up to 90 lines are displayed.
      a. *sar −u 1 80*
3) To redirect this and save the output into a text file in the current directory

a. *sar -u 1 80 > cpu.tsv*

X. CREATE SECURITY ASSOCIATIONS WITH IPSEC-TOOLS

Guide: https://help.ubuntu.com/community/IPSecHowTo

http://www.ipsec-howto.org/x299.html

1) Install the ipsec-tools 0.7.2(see COMPILING EXTRA PACKAGES FOR OVERO IMAGE)
2) To generate pseudo-random pre-shared keys for security associations:
    a. For SHA-256 and AES 256-bit keys:
        i. dd if=/dev/random count=32 bs=1| xxd –ps
3) Edit /etc/ipsec-tools.conf, to add/remove security associations
4) To apply changes
    a. sudo setkey –f /etc/ipsec-tools.conf
5) To view active security associations
    a. sudo setkey –D
6) To view active security policies
    a. sudo setkey –DP

Appendix D: Wiring diagram of Power Monitor

DB-25P (Male Plug Front View)

DB-25S (Female Socket Front View)

Wagan Tech

+14V
Gnd

MTR72DAUL-1250A
AC to DC Converter
14VDC @ 5A

Battery +
Battery –
Battery – Charge

iRobot Create 4400

DB-25 Fem
Serial + (1)
Serial – (2)
Battery + (10)
Battery – (25)

DB-25 Male
(1) Serial +
(2) Serial –
(10) +15V
(25) Gnd

Gumstix Overo Earth

Ext Power
+5V
Gnd

R1: 0.1Ω
10W

CUI INC

+5V
Gnd

EPS05100
Regulated
AC to DC Converter
5VDC @ 1A

USB

Power Monitor
Dell Latitude E6510
Windows 7 SP1 64-bit
NI LabVIEW 9.0.1 32-bit

+/- 10V Diff Mode

+/- 1V Diff Mode

USB

(1) GND	P0.0 (17)
(2) AI 0 (AI 0+)	P0.1 (18)
(3) AI 4 (AI 0-)	P0.2 (19)
(4) GND	P0.3 (20)
(5) AI 1 (AI 1+)	P0.4 (21)
(6) AI 5 (AI 1-)	P0.5 (22)
(7) GND	P0.6 (23)
(8) AI 2	P0.7 (24)
(9) AI 6	P1.0 (25)
(10) GND	P1.1 (26)
(11) AI 3	P1.2 (27)
(12) AI 7	P1.3 (28)
(13) GND	PFI 0 (29)
(14) AO 0	+2.5V (30)
(15) AO 1	+5V (31)
(16) GND	GND (32)

NI
USB-6008 DAQ

Note: Per the NI USB-6008 User
Guide, no AI channel can have
more than +10V applied to it.

AI Input Impedence: 144kΩ

Appendix F: Data Tables

Factor Levels: Defense=None; Exploit=Passive Sniffing

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	3.462	2.114	3784	355726	35.5726
2	1	3.310	2.145	3790	356260	35.626
3	1	3.028	2.150	3773	354720	35.472
4	1	3.267	2.145	3781	355478	35.5478
5	1	3.256	2.147	3820	359080	35.908

Factor Levels: Defense=None; Exploit=ARP Cache Poisoning

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	3.824	2.146	7469	705026	70.5026
2	1	3.757	2.152	7438	701740	70.174
3	1	3.205	2.144	7470	705092	70.5092
4	1	3.116	2.151	7406	699228	69.9228
5	1	5.055	2.122	7473	705622	70.5622

Factor Levels: Defense=None; Exploit=TCP Connection Hijacking

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	96.296	2.319	1361730	89874756	8987.4756
2	0	94.741	2.350	1369435	90383510	9038.351
3	1	97.007	2.323	1348911	89028798	8902.8798
4	1	94.829	2.350	1389899	91734298	9173.4298
5	1	95.408	2.321	1362125	89900814	8990.0814

Factor Levels: Defense=None; Exploit=TCP Reset

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	3.039	2.123	17	1402	0.1402
2	1	2.965	2.149	15	1190	0.119
3	1	3.132	2.157	25	2130	0.213
4	1	2.680	2.153	15	1190	0.119
5	1	2.711	2.158	15	1190	0.119

Factor Levels: Defense=None; Exploit=TCP Connection Flooding

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	41.420	2.228	13702	1057564	105.7564
2	1	39.499	2.206	13195	1027288	102.7288
3	1	40.406	2.234	13566	1047996	104.7996
4	1	40.471	2.233	13242	1028818	102.8818
5	1	38.990	2.211	13369	1038380	103.838

Factor Levels: Defense=IPsec AH; Exploit=Passive Sniffing

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	1	3.541	2.140	3800	448400	44.84
2	1	3.511	2.148	3791	447402	44.7402
3	1	3.201	2.151	3820	450760	45.076
4	1	3.580	2.149	3781	446130	44.613
5	1	3.145	2.120	3805	448955	44.8955

Factor Levels: Defense=IPsec AH; Exploit=ARP Cache Poisoning

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.961	2.154	48	4832	0.4832
2	0	2.794	2.123	44	4592	0.4592
3	0	2.765	2.157	52	5072	0.5072
4	0	2.805	2.151	50	4952	0.4952
5	0	3.150	2.155	52	5072	0.5072

Factor Levels: Defense=IPsec AH; Exploit=TCP Connection Hijacking

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.532	2.145	5487	626951	62.6951
2	0	3.537	2.150	7588	849844	84.9844
3	0	3.702	2.116	7612	852648	85.2648
4	0	3.358	2.145	7583	849222	84.9222
5	0	3.375	2.147	7558	846684	84.6684

Factor Levels: Defense=IPsec AH; Exploit=TCP Reset

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.296	2.145	8233	792986	79.2986
2	0	3.365	2.145	8275	797062	79.7062
3	0	3.269	2.145	8186	788508	78.8508
4	0	3.355	2.113	8212	790936	79.0936
5	0	3.343	2.143	8240	793696	79.3696

Factor Levels: Defense=IPsec AH; Exploit=TCP Connection Flooding

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.571	2.119	10338	839900	83.99
2	0	3.447	2.147	10394	843660	84.366
3	0	3.422	2.147	10355	841350	84.135
4	0	3.584	2.150	10322	839170	83.917
5	0	3.755	2.118	10333	838762	83.8762

Factor Levels: Defense=IPsec ESP; Exploit=Passive Sniffing

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.552	2.161	3782	491692	49.1692
2	0	3.939	2.164	3770	490100	49.01
3	0	3.226	2.167	3785	491994	49.1994
4	0	3.153	2.165	3800	494000	49.4
5	0	3.197	2.138	3815	495622	49.5622

Factor Levels: Defense=IPsec ESP; Exploit=ARP Cache Poisoning

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	5.754	2.166	7580	983168	98.3168
2	0	5.393	2.167	7671	995126	99.5126
3	0	3.664	2.169	7557	980618	98.0618
4	0	4.024	2.134	7599	986190	98.619
5	0	3.865	2.171	7591	985054	98.5054

Factor Levels: Defense=IPsec ESP; Exploit=TCP Connection Hijacking

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.271	2.169	3801	494186	49.4186
2	0	3.221	2.170	3778	491020	49.102
3	0	3.447	2.137	3790	492700	49.27
4	0	3.068	2.142	3820	496600	49.66
5	0	3.068	2.138	3767	489718	48.9718

Factor Levels: Defense=IPsec ESP; Exploit=TCP Reset

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.688	2.139	3791	492886	49.2886
2	0	3.248	2.136	3792	492992	49.2992
3	0	3.187	2.138	3777	491018	49.1018
4	0	3.316	2.140	3771	490286	49.0286
5	0	4.143	2.139	3772	490392	49.0392

Factor Levels: Defense=IPsec ESP; Exploit=TCP Connection Flooding

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.994	2.140	10838	916980	91.698
2	0	3.562	2.138	10633	901180	90.118
3	0	3.732	2.135	10704	908646	90.8646
4	0	4.843	2.168	10693	908234	90.8234
5	0	3.700	2.165	10691	905654	90.5654

Factor Levels: Defense=IPsec AH+ESP; Exploit=Passive Sniffing

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.670	2.152	3798	584692	58.4692
2	0	3.923	2.124	3802	585540	58.554
3	0	3.935	2.154	3777	581666	58.1666
4	0	3.684	2.156	3801	585410	58.541
5	0	3.701	2.153	3806	586156	58.6156

Factor Levels: Defense=IPsec AH+ESP; Exploit=ARP Cache Poisoning

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.874	2.123	7544	1159768	115.9768
2	0	4.138	2.152	7613	1170188	117.0188
3	0	3.937	2.153	7584	1165800	116.58
4	0	4.087	2.127	7588	1166160	116.616
5	0	4.280	2.125	7569	1163418	116.3418

Factor Levels: Defense=IPsec AH+ESP; Exploit=TCP Connection Hijacking

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	4.135	2.155	3811	586950	58.695
2	0	4.132	2.149	3761	579170	57.917
3	0	4.132	2.150	3801	585410	58.541
4	0	3.860	2.151	3801	585410	58.541
5	0	3.972	2.151	3791	583870	58.387

Factor Levels: Defense=IPsec AH+ESP; Exploit=TCP Reset

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	3.732	2.119	3781	582330	58.233
2	0	4.311	2.149	3771	580790	58.079
3	0	3.756	2.153	3762	579380	57.938
4	0	3.630	2.153	3796	584584	58.4584
5	0	3.878	2.147	3786	583044	58.3044

Factor Levels: Defense=IPsec AH+ESP; Exploit=TCP Connection Flooding

Replication	Exploit Outcome (Successes=1; Failure=0)	Average CPU Usage (%)	Average Power (W)	Packets (packets)	Network Load (Bytes)	Network Load (Kbps)
1	0	4.143	2.127	10241	969539	96.9539
2	0	4.172	2.149	10409	980870	98.087
3	0	5.253	2.120	10215	967694	96.7694
4	0	3.952	2.152	10461	985456	98.5456
5	0	4.099	2.123	10294	972960	97.296

Bibliography

[ANL01]   Aura, T., Nikander, P., and Leiwo, J. 2001. "DOS-resistant authentication with client puzzles." Springer: Security Protocols. Retrieved 21 March 2012. http://www.tcs.hut.fi/old/papers/aura/aura-nikander-leiwo-protocols00.pdf

[AVT04]   Argyroudis, P., Verma, R., Tewari, H., and O'Mahony, D. 2004. "Performance analysis of cryptographic protocols on handheld devices." *IEEE.* Third IEEE International Symposium on Network Computing Applications, 2004. pg. 169-174

[BSP95]   Bakhtiari, S., Safavi-Naini, R., and Pieprzyk, J. 1995. "Cryptographic hash functions: A survey." *Citeseer.* Retrieved 21 March 2012. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.8428&rep=rep1&type=pdf

[Ber11]   Bernstein, D. 2011. "SYN Cookies." Retrieved 19 July 2011. http://cr.yp.to/syncookies.html

[Bel96]   Bellovin, S. 1996. "Problem areas for the IP security protocols." *Proceedings of the Sixth Usenix Unix Security Symposium.* pg. 1-16

[Bhi96]   Bhimani, A. 1996. Securing the commercial Internet. *ACM.* Communications of the ACM Volume 39 Issue 6. pg. 29-35

[BBH11]   Bittau, A., Boneh, D., Hamburg, M. 8 September 2011. "Cryptographic protection of TCP Streams (tcpcrypt)." *Internet Engineering Task Force.* Retrieved 1 December 2011. http://tools.ietf.org/html/draft-bittau-tcp-crypt-01

[BHH10]   Bittau, A., Hamburg, M., Handley, M., Mazieres, D., Boneh, D. 2010. "The case for ubiquitous transport-level encryption." *USENIX Security Symposium.*

[Bit12]   Bittau, A. 2012. "About us." Retrieved 30 January 2012. http://tcpcrypt.org/aboutus.php

[BlC06]   Black, J., Cochran, M. 2006. *Fast Software Encryption.* "A study of the MD5 attacks: Insights and improvements." *Springer.* pg 262-277.

[CDN00]   Caldera, J., De-Niz, D., and Nakagawa, J. 2000. "Performance analysis of IPSec and IKE for mobile IP on wireless environments." *Information Networking Institute, Carnegie Mellon University.*

[Cal07]   Callas, J. November 2007. "OpenPGP Message Format" RFC 4880 Retrieved 8 July 2011. http://www.ietf.org/rfc/rfc4880.txt

[Cer98]   CERT. 1998. "Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks." *Carnegie Mellon.* Retrieved 19 July 2011. http://www.cert.org/advisories/CA-1998-01.html

[Cha04]   Chau, H.   2004. "Network Security – Defense Against DoS/DDoS Attacks."
             shadowspy.free.fr.   Retrieved 21 March 2012.
             http://shadowspy.free.fr/ebooks/security-hacking/defense_ddos.pdf

[Dan10]   Dantzig, G. 2010. "The Nature of Mathematical Programming." *Mathematical
             Programming Glossary, INFORMS Computing Society.* Retrieved 13 December
             2011. http://glossary.computing.society.informs.org/index.php?page=nature.html

[Dai09]   Dai, W.   2009. "Crypto++ 5.6.0 Benchmark." Retrieved 24 January 2012.
             http://www.cryptopp.com/benchmarks.html

[Del11]   *Dell.* 2011. Latitude D630 Laptop. Retrieved 3 November 2012.
             http://www.dell.com/us/dfb/p/latitude-d630/pd

[Die08]   Dierks, T. August 2008. The Transport Layer Security (TLS) Protocol Version
             1.2 RFC 5246. Retrieved 8 July 2011. http://tools.ietf.org/html/rfc5246

[EMS02]   Elkeelany, O., Matalgah, M., Sheikh, K., Thaker, M., Chaudhry, G., Medhi, D.,
             and Qaddour, J. 2002. "Performance analysis of IPSec protocol: encryption and
             authentication" *IEEE.* IEEE International Conference on Communications, 2002
             Volume 2. pg. 1164-1168

[Fre00]   *The FreeBSD Project.* 29 December 2000. "External Data Representation
             Standard: Protocol Specification." Retrieved May 26, 2011.
             http://docs.freebsd.org/44doc/psd/25.xdrrfc/paper.pdf

[Fri05]   Friedl, S. 24 August 2005. "An Illustrated Guide to IPsec". Retrieved 1
             December 2011.   http://www.unixwiz.net/techtips/iguide-ipsec.html

[Ger00]   Gerck, E.   2000. "Overview of Certification Systems: X. 509, PKIX, CA, PGP &
             SKIP." *The Bell Volume 1.* pg.8

[GSV00]   Gerkley, B., Stoy, K., and Vaughan, R. 22 November 2000. "Player Robot Server
             Version 0.7.4a User Manual." *USC Robotics Labs.* Retrieved 21 March 2012.
             http://wotan.liu.edu/docis/lib/musl/rclis/dbl/cogrfo/(1994)13%253A3%253C33%
             253AAPFTRD%253E/robotics.usc.edu%252F~gerkey%252Fresearch%252Ffinal_
             papers%252Fplayer.pdf

[GVS01]   Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G., and Mataric, M.
             2001. "Most valuable player: A robot device server for distributed control." *IEEE.*
             2001 IEEE/RSJ International Conference on Intelligent Robots and Systems
             Volume 3. pg. 1226-1231

[GuT95]   Guynes C, and Thorn, R. 1995. "Network security in a client/server
             environment." *ACM.* SIGSAC Review Volume 13 Issue 2. pg. 6-12

[HeB96]   Heberlein, L., and Bishop, M. 1996. "Attack class: Address spoofing." *Proceeding
             of the 19th National Information Systems Security Conference.* pg. 371-377

[Ips11]   *IPsec HOWTO.* 2011. "Theory: What is IPsec?" Retrieved 15 July 2011.
             http://www.ipsec-howto.org/x202.html

[Iro11]   *iRobot.* 9 Aug 2011. "iRobot Create® Programmable Robot." Retrieved 9 Aug 2011. http://store.irobot.com/shop/index.jsp?categoryId=3311368

[JHE99]   Jing, J., Helal, A., and Elmagarmid, A. 1999. "Client-server computing in mobile environments." *ACM.* Computing Surveys (CSUR) Volume 31 Issue 2. pg. 117-157

[Jon95]   Joncheray, L. 1995. "A simple active attack against TCP." *Proceedings of the Fifth Usenix Unix Security Symposium.* pg. 7-19

[Kim07]   Kim, H. 2007. IP Network Security: IPsec (Internet Protocol Security). Retrieved 8 July 2011. http://www.ibluemojo.com/contents/IP_Network_Security.pdf

[KeS05]   Kent, S., and Seo, K. December 2005. Security Architecture for the Internet Protocol - RFC 4301. Retrieved 8 July 2011. http://trac.tools.ietf.org/html/rfc4301

[Ken05]   Kent, S. December 2005. IP Authentication Header - RFC 4302. Retrieved 8 July 2011. http://trac.tools.ietf.org/html/rfc4302

[Ken05a]   Kent, S. December 2005. IP Encapsulating Security Payload(ESP) - RFC 4303. Retrieved 8 July 2011. http://trac.tools.ietf.org/html/rfc4303

[KuR10]   Kurose, J, and Ross, K. 2010. *Computer networking 5th edition.* New York City: Addison-Wesley. pg. 716-718

[KuR10a]   Kurose, J, and Ross, K. 2010. *Computer networking 5th edition.* New York City: Addison-Wesley. pg. 707-709

[Len11]   *Lenovo.* 2011. "Drivers and software – Thinkpad T43, T43P." Retrieved 3 November 2011. http://support.lenovo.com/en_US/research/hints-or-tips/detail.page?&LegacyDocID=MIGR-58597

[MDR05]   Mirkovic, J., Dietrich, S., and Reiher, P. 2005. *Internet denial of service: attack and defense mechanisms.* Upper Saddle River, NJ: Prentice Hall.

[Nis03]   *NIST.* June 2003. "National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information." Retrieved 30 January 2012. http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf

[Nis08]   *NIST.* 10 December 2008. "Tentative Timeline of Development of New Hash Functions." *U.S. Department of Commerce.* Retrieved 30 January 2012. http://csrc.nist.gov/groups/ST/hash/timeline.html

[Obf12]   Obfuscated TCP. 2012. *eNotes.com Inc.* Retrieved 17 Jan 2012. http://www.enotes.com/topic/Obfuscated_TCP

[OrV03]   Ornaghi, A., and Valleri, M. 2003. "Man in the middle attacks Demos." *Blackhat [Online Document].* Retrieved 21 March 2012. https://blackhat.com/presentations/bh-usa-03/bh-us-03-ornaghi-valleri.pdf

[Ope09]   OpenBSD. 2009. OpenSSH Homepage. Retrieved 8 July 2011.
            http://www.openssh.org/

[Owe10]   Owen, J. 16 April 2010. "How to Use Player/Stage 2nd Edition." Retrieved 11 July
            2011. http://www-users.cs.york.ac.uk/~jowen/player/playerstage-tutorial-
            manual.pdf

[Pla11]   Player Project. 25 May 2011. "libplayertcp Developer Manual." Retrieved 18 July
            2011. http://playerstage.sourceforge.net/doc/Player-
            3.0.2/player/group__libplayertcp.html

[Phi07]   Philip, R. 2007. "Securing wireless networks from ARP cache poisoning." *Citeseer.*
            Retrieved 21 March 2012.
            http://www.cs.sjsu.edu/faculty/stamp/students/Roney298report.pdf

[Gum11]   *Gumstix Inc.* 2011. "Product Change Notices." Retrieved 3 November 2012.
            http://www.gumstix.org/hardware-design/product-changes-known-issues-a-
            eol/66-product-change-notices/196-2011-product-change-notices.html

[PaY06]   Paterson, K., Yau, A. 2006. "Cryptography in theory and practice: The case of
            encryption in IPsec." *Advances in Cryptology-EUROCRYPT 2006.* pg. 12-29.

[Rsa11]   *RSA Laboratories.* 2011. "What is SSL?" Retrieved 8 July 2011.
            http://www.rsa.com/rsalabs/node.asp?id=2293

[Rsa11a]   *RSA Laboratories.* 2011. "Has DES been broken?" Retrieved 2 Dec 2011.
            http://www.rsa.com/rsalabs/node.asp?id=2227

[Sca11]   *Scapy.* 2011. "About Scapy." Retrieved 9 December 2011.
            http://www.secdev.org/projects/scapy/

[Sko06]   Skoudis, E. 2006. *Counter hack reloaded.* Upper Saddle River, NJ: Prentice Hall.
            pg. 542-543

[TrW06]   Trappe, W., and Washington, L. 2006. *Introduction to Cryptography with
            Coding Theory Second Edition.* Upper Saddle River, NJ: Prentice Hall.

[Tex11]   *Texas Instruments.* 7 January 2011. "OMAP3530 Power Estimation
            Spreadsheet." Retrieved 2 February 2012.
            http://processors.wiki.ti.com/index.php/OMAP3530_Power_Estimation_Spread
            sheet

[WYY05]   Wang, X., Yin, Y., Yu, H. 2005. "Finding collisions in the full SHA-1." *Advances
            in Crytology—CRYPTO 2005.* pg. 17-36

[Wat04]   Watson, P. 2004. "Slipping in the Window: TCP Reset Attacks." *CanSecWest.*
            Retreived 21 March 2012.
            http://bandwidthco.com/whitepapers/netforensics/tcpip/TCP%20Reset%20Atta
            cks.pdf

[Wha01]   Whalen, S. 2001. "An introduction to ARP spoofing." *Node99 [Online Document].* Retrieved 21 March 2012.
http://www.leetupload.com/database/Misc/Papers/arp_spoofing_slides.pdf

[Wir11]   *Wireshark Foundation.* 2011. "About Wireshark." Retrieved 9 December 2011.
http://www.wireshark.org/about.html

[YIS07]   Yatagai, T., Isohara, T., Sasase I. 2007. "Detection of HTTP-GET flood attack based on analysis of page access behavior." *IEEE. IEEE* Pacific Rim Conference on Communications, Computers and Signal Processing, 2007. pg 232-235.

1. REPORT DATE (DD-MM-YYYY) 14-06-2012	2. REPORT TYPE Master's Thesis	3. DATES COVERED (From – To) March 2011 – June 2012

TITLE AND SUBTITLE Vulnerability Analysis of the Player Command and Control Protocol	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Hagen, John T., Civ, USAF	5d. PROJECT NUMBER 12G289
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENG) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCO/ENG/12-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Labs AFRL/RYWC 2241 Avionics Circle Wright-Patterson AFB OH 45433-7334 POC: Juan Carbonell, Chief (937) 798-8140 juan.carbonell@wpafb.af.mil	10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RYWC
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
    DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
The Player project is an open-source effort providing a control interface specification and software framework for abstracting robot hardware. This research presents five exploits that compromise vulnerabilities in Player's command and control protocol. The attacks exploit weaknesses in the ARP, IP, TCP and Player protocols to compromise the confidentially, integrity, and availability of communication between a Player client and server. The attacks assume a laptop is connected in promiscuous mode to the same Ethernet hub as the client and server in order to sniff all network traffic between them. This work also demonstrates that Internet Protocol Security (IPsec) is capable of mitigating the vulnerabilities discovered in Player's command and control protocol. Experimental results show that all five exploits are successful when Player communication is unprotected but are defeated when IPsec Authentication Header (AH) and Encapsulating Security Protocol (ESP) are deployed together (AH+ESP) in transport mode. A cost function is defined to synthesize three distinct scalar costs (exploit success, CPU utilization, and network load) into a single scalar output that can be used to compare the different defense protocols provided by IPsec. Results from this cost function show that in a scenario when exploits are likely, IPsec AH+ESP is the preferred defense protocol because of its relatively low CPU and network overhead and ability to defeat the exploits implemented in this research by authenticating and encrypting the transport and application layers. Performance data reveals that for the Overo Earth embedded system running a TI OMAP3530 processor at 720MHz, IPsec AH+ESP increases CPU utilization by 0.52% and the network load by 22.9Kbps (64.3% increase).

15. SUBJECT TERMS
network attack, security, exploit, IPsec, performance, cost, Player

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Dr. Barry Mullins, Civ, USAF ADVISOR
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	126	19b. TELEPHONE NUMBER (Include area code) (937) 785-3636, ext 7979 barry.mullins@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18